INSTITUTO
SUPERIOR
TÉCNICO

**UNIVERSIDADE TÉCNICA DE LISBOA**

**INSTITUTO SUPERIOR TÉCNICO**

# Statically Proving Behavioural Properties in the π-calculus via Dependency Analysis

Maxime Emile Gamboni
(Master's Degree in Computer Science)

Thesis specifically prepared to obtain the PhD Degree in Mathematics

Supervisor: Doctor António Maria Lobo César Alarcão Ravara
Co-Supervisor: Doctor Carlos Manuel Costa Lourenço Caleiro

December 2010

**Título** Determinação Estática de Propriedades Comportamentais no Cálculo $\pi$, usando Análise de Dependências

**Nome** Maxime Gamboni

**Doutoramento em** Matemática

**Orientador** António Ravara

**Co-orientador** Carlos Caleiro

**Resumo** Nesta tese apresento um mecanismo semântico genérico e um sistema de tipos provado correcto para analisar propriedades comportamentais do cálculo-$\pi$. Além de propriedades de animação tais como actividade (uma generalisação da receptividade), alcance e terminação, o mecanismo também suporta a análise de propriedades de segurança tais como determinismo e isolamento.

A análise de dependências é uma parte central deste mecanismo, funcionando com *declarações de dependências* descrevendo propriedades de um processo condicionadas por recursos esperado de processos exteriores. As declarações de dependência são usadas como partes elementares de *declarações comportamentais*, declarações lógicas descrevendo a negociação de recursos entre um processo e o seu ambiente. A análise de dependências traz uma poderosa propriedade de *composicionalidade*: compondo elementos pré-analisados (tipados), o tipo do processo resultante pode ser directamente obtido a partir dos tipos dos elementos, sem precisar de uma análise separada do processo completo. As declarações comportamentais também integram primitivas para selecção (escolhas feitas por um processo) e ramificação (escolhas oferecidas por um processo).

O sistema de tipos, parameterizado com *regras elementares* dando a essência das propriedades pretendidas e de *tipos de canais* dando o protocolo esperado em canais de comunicação, constroi uma declaração comportamental automaticamente, analisando os processos.

**Palavras-chave** cálculo-$\pi$, propriedades de animação, propriedades de segurança, sistemas de tipo genéricos, actividade, codificações, escolha.

**Title** Statically Proving Behavioural Properties in the $\pi$-calculus via Dependency Analysis

**Abstract** In this thesis I present a generic semantic framework and sound type system suited for analysing a wide variety of behavioural properties in $\pi$-calculus processes, both liveness properties such as activeness (a generalisation of receptiveness), termination and reachability, and safety properties such as determinism and isolation.

Dependency analysis is a central ingredient of this framework, implemented by *dependency statements* describing process properties conditional on some resources to be provided by third-party processes. Dependency statements are used as elementary ingredients of *behavioural statements*, logical statements that precisely characterise negotiation of resources between a process and its environment. Dependency analysis provides this framework with powerful *compositionality*: when arranging pre-analysed (typed) components together, the resulting process' type can be directly derived from those of the components, with no need to re-analyse the entire process. Behavioural statements also integrate primitives for describing *selection* (choices made by a process) and *branching* (choices offered by a process).

The type system, parametrised with *elementary rules* giving the essence of the desired properties and *channel types* giving communication protocols to be used on communication channels, automatically constructs a behavioural statement by automatically analysing processes.

**Keywords** $\pi$-calculus, liveness properties, safety properties, generic type systems, activeness, encodings, choice.

## Acknowledgements

I wish to thank António, Kohei, Lucia, Nobuko, Simon, Uwe for their continued and repeated help, advice and support, and Carlos for agreeing on very short notice to act as my co-supervisor.

Thank you to my parents for financially supporting most of my expenses during my studies, and for making me keep going when it looked like proving theorems from this thesis would never be complete.

“This work is partially supported by SQIG — Instituto de Telecomunicações and IST, Portugal, by Fundação para a Ciência e a Tecnologia, as well as the EU FET-GC project Sensoria (IST-2005-16004).”

# Contents

# Chapter 1

# Introduction

This thesis is about the use of *dependency analysis* to study the behaviour of mobile processes.

## 1.1 Mobile Processes

Similarly to the $\lambda$-calculus being a foundational framework to study sequential functional programming, *process calculi* are formalisms providing theory to reason about *open*, *concurrent*, *distributed* and *mobile* systems. A distributed algorithm is one spanning many information systems like sensor networks, (physically) mobile agents, computers in a peer-to-peer network, or just processes in a single computer. In order to precisely describe distributed processes, one needs a *parallel composition* connective "$|$", which, when applied to a number of processes, indicates that they can run independently of each other. For instance $P_1 \mid P_2 \mid P_3 \mid P_4$ represents a process with four independent components that may be running on four different systems.

As the components may be located in different places, they can't share information (like two threads of a program may access a common memory heap, or like two of a computer would share a common memory), but must instead rely on explicit *communication* to exchange information. Some calculi such as Linda [CG90] use a broadcast model in which all participants of an algorithm may access a common "tuple space" that serves as a unique communication medium. *Nominal calculi* such as the Calculus of Communicating Systems or CCS [Mil80] use *named channels.* In this thesis we will focus on nominal calculi.

Communication is implemented with two primitives, input $a$ and output $\bar{a}$, where $a$ is the name of the communication channel. The *prefix* or sequence connective "." is used for synchronisation: $a.P$ is a program that waits for a signal on channel $a$, then executes $P$, and $\bar{a}.P$ is a program that sends a signal on channel $a$, then whenever it is received, proceeds with $P$. It should be noted that a message can only be received by a single receiver, so $a.P \mid a.Q \mid \bar{a}$ is a program that *non-deterministically* runs one of $P$ and $Q$, leaving the other one deadlocked (assuming $a$ doesn't appear elsewhere in $P$ or $Q$).

The $\pi$-calculus [MPW92, SW01] goes beyond synchronisation and permits explicit information transfer by letting outputs carry values ($\overline{a}\langle x_1, x_2, \ldots\rangle$), abbreviated $\overline{a}\langle\tilde{x}\rangle$, and inputs carry variables ($a(y_1, y_2, \ldots)$), much like function calls and function declarations in many programming languages.

Many extensions and variants of process calculi have been developed, to cover a wide variety of purposes, such as the *spi-calculus* [AG97] having primitives for encryption and decryption, a Distributed $\pi$-calculus [Hen07], having explicit representation of computation *sites* whereby agents can *move* from one site to another and can only communicate with each other when they are in the same site. The Higher-Order $\pi$-calculus HO$\pi$ [San93], allows transmitting entire processes over channels, much like web browsers download applets from servers and execute them. It is also common to extend the basic $\pi$-calculus with primitive data types and operators. Finally, actual programming languages have been developed. Pict [PT00] is an experimental implementation closely matching the $\pi$-calculus, while TyCO [Vas94] is an *object oriented* extension.

## 1.2 Equivalences and Encodings

With such a wealth of process calculi to choose from, two questions arise: do these extensions merely bring convenience and shortcuts to the programmers (and researchers), or do they fundamentally increase the expressive power of the language? Should the answer be the latter (i.e. the languages are deemed to be equally expressive), can programs, as well as theoretical results, applying to one calculi be "translated" into corresponding ones into another one?

A key step in answering these questions consists of writing an *encoding* from one calculus to another. Should faithful encodings exist in both directions between two calculi, they can reasonably be considered to have equivalent expressive power.

I started my journey in the world of process calculi seven years ago [GNR04], with this very question: do TyCO and Sangiorgi's asynchronous $\pi$-calculus with *variants* [San98] have the same expressive power?

Two calculi are equally expressive if there exist encodings $[\![\,\cdot\,]\!]$ from one to the other such that [Nes00]

- the encodings are *distributed* or *compositional* (e.g. $[\![\, P \,|\, Q \,]\!] = [\![\, P \,]\!] \mid [\![\, Q \,]\!]$)
- the encodings are fully abstract (e.g. $P \approx Q \iff [\![\, P \,]\!] \approx' [\![\, Q \,]\!]$ where $\approx'$ is some "suitable" equivalence relation.)

The interested reader may want to read [Par08] for an extensive survey of techniques for assessing relative (and absolute) expressiveness of process calculi.

One goal of an encoding is to permit encoded processes to interact with arbitrary processes of the target calculus that were not necessarily obtained through the encoding.

A question immediately arises: what is a suitable equivalence relation for this situation? Having $(\approx') = (\approx)$ is usually not feasible because the equivalence relation typically needs to hide artifacts introduced by the encoding. For instance we want to be able to encode in polyadic $\pi$-calculus (that only allows transmitting names) expressions such as $\overline{a}\langle\xi\rangle$ where $\xi$ is a complex structure that may be of unbounded size (such as a list). The only way this can be achieved is by first sending a name containing a pointer to the data to be transmitted, and then transmitting the data small parts at a time. After the encoding, data is no longer a primitive entity, and operations done on data are no longer atomic, so usual equivalence relations may be able to distinguish normally indistinguishable processes by violating the semantics normally associated with data, thereby breaking the full abstraction requirement.

One approach for defining a "suitable equivalence" is to build on barbed congruence [SW01] but restricting which contexts are allowed:

**Definition 1.2.1 (Post-Encoding Equivalence)** *Two encoded processes $P$ and $Q$ are* equivalent *(written $P \approx' Q$) if for all processes $R$* well-behaved with respect to *both $P$ and $Q$, $(P \mid R) \overset{\bullet}{\approx} (Q \mid R)$ where $\overset{\bullet}{\approx}$ is the usual barbed bisimulation relation.*

## 1.3 Behavioural Properties

The question we now need to answer is "what is a *well-behaved* process (with respect to some encoded process $P$)?". To see why we need to restrict which processes may interact with an encoded process, consider the two following processes:

> $P$ receives a value $v$ on channel $a$, sends a signal on channel $s$ and then decodes value $v$ (discarding the result).
>
> $Q$ receives a value $v$ on channel $a$, then decodes value $v$ (discarding the result) and finally sends a signal on channel $s$.

Assuming value decoding is immediate, those two processes are indistinguishable for an external observer as both receive a value on $a$ and then send a signal on $s$ (in case value decoding takes a measurable time, they can be made bisimilar again by inserting silent actions of corresponding lengths at the corresponding places).

However the encoded forms of these processes are, respectively, as follows[1]:

> $[\![ P ]\!]$ receives on channel $a$ a name $u$ holding an encoding of value $v$, then sends a signal on channel $s$, and finally sends a decoding request on channel $u$, discarding the reply.
>
> $[\![ Q ]\!]$ receives on channel $a$ a name $u$ holding an encoding of value $v$, then sends a decoding request on channel $u$. *After receiving the reply*, it sends a signal on channel $s$.

Now these two processes can be distinguished by a process $R$ sending a (private) name $u$ and ignoring any decoding requests: $[\![ P ]\!]$ will send the success signal but $[\![ Q ]\!]$ will not, as it will be blocked waiting for a reply to its decoding request.

More generally, test processes $R$ must provide the "FAIR Semantics"[2] on channels holding value encodings:

1. *Functionality*: Once sent, the transmitted data is fully determined, and does not change from one access to the other. This property is covered in Section 8.2, where it is called "determinism".

[1]Although there are other ways to encode these processes, all exhibit a similar difficulty.
[2]All credit goes to Uwe Nestmann for the name.

2. *Activeness*: The data can be accessed by the receiver as many times as it wants.

3. *Isolation*: The sender has no way of knowing when and how many times the data is decoded by the receiver.

4. *Responsiveness*: A decoding of the data always succeeds and terminates after a finite time.

I spent the first years of my PhD developing semantics and type systems guaranteeing those properties as a whole, but soon enough it became obvious that, as rightfully pointed out by conference reviewers at the time, this work deserves to be more modular (to permit cherry-picking desired properties rather than the monolithic all-or-nothing characterisation) and generic (to be easily generalisable for other properties). Although encodings were the initial motivation for this research, behavioural type systems are also useful for verifying algorithms, for instance responsiveness (Section 4.8), termination (Section 8.4) and deadlock-freedom (Section 8.5) are important properties for long-running applications such as web-servers or for algorithms running on devices difficult to service, like sensor networks or space probes.

In this thesis we will not work directly on encodings but rather on characterisation of behavioural properties in mobile processes. We will also focus on the $\pi$-calculus to fix the notation, as most of the material can be painlessly applied to other process calculi.

It must be noted at this point that the requirement that encodings should be distributed is not as absolute as previously thought. Refer to [BPV05] for an encoding of $\pi$-calculus with data into the base $\pi$-calculus, that is merely *weakly compositional* [Par08], and uses a central value server to avoid the above difficulties (neither encoded processes nor test processes may carry encoded values, instead they register them into the value server, that guarantees all “FAIRness” properties by construction).

## 1.4 A Process and its Environment

Parallel composition allows putting together several pre-written components to build a more complex application. It is therefore natural that, when reasoning about such a component, whether it is to specify its desired behaviour or verify an actual implementation, one should put an emphasis on interaction with unspecified or under-specified third-party components.

This theme comes in various flavours in the $\pi$-calculus literature.

This is for example found in equivalence relations used on mobile processes. The *barbed congruence relation* [SW01] states that two processes $P$ and $Q$ are equivalent if, no matter in which *context* $C[\cdot]$ they are put, $P$ and $Q$ provide the same barbs (offers or attempts at communication with the environment).

Another instance is the use of *labelled* transitions. The transition $a.P \xrightarrow{a} P$ implicitly makes the assumption that the process found a communication partner $\bar{a}$, not found inside the $a.P$ notation, but rather in an unspecified third-party process. Labelled transitions are central to equivalences such as bisimulation or trace equivalence.

One consequence is that, when specifying, designing, writing, analysing and verifying a process, one should keep in mind the fact that the process may be running together and interacting with other unspecified processes. We shall the use the words *local process* when referring to a process being studied, while *remote* or *environment* refers to the composition of all those other processes.

Therefore, when writing a software component in a mobile calculus it quickly becomes important to specify in what ways the environment is permitted to interact with it. The restriction connective $(\boldsymbol{\nu} a)$ can be used as a crude first step in that direction, in that it permits specifying that a channel should be *private* or *internal* to a process and the environment is not permitted to interfere with it (it of course has other uses, such as the creation of one private channel for every request to a replicated server). Finer limitations not expressed in the process syntax would include permitting access to only one side of a channel ("third-party processes may output on $a$, but not input on it"), limiting the number of uses ("third-party processes must do exactly one output on $a$"), etc.

This last example is particularly interesting in that it changes when the process interacts with the environment: if it goes through a labelled input transition $\xrightarrow{a}$, it is assumed that this transition corresponds to an output that was sent on $a$ from a third-party process, and therefore the constraint on the environment must be changed to ("third-party process must *not* do any output on $a$").

## 1.5 Choice

In process calculi, processes can make and communicate *choices*, a fundamental component of data representation (where a piece of data matches one of a set of patterns), of object-oriented style programming (where a call matches one method out of a set) or session-based programming (during a conversation between a client and a server, both sides are at times permitted to drive the protocol one way or another). We shall use *branching* and *selection* to capture properties in process constructs necessary for such usage patterns.

For example, Milner's encoding of Boolean values [Mil93] represents Boolean values as receivers on two parameter channels: $\mathsf{True}$ replies to queries with a signal on the first parameter $(!\,b(tf).\bar{t})$ and $\mathsf{False}$ with a signal on the second parameter $(!\,b(tf).\bar{f})$. We say those two processes are instances of *selection* (sometimes called internal choice in the literature) because they pick a specific behaviour out of a set of mutually exclusive permissions, by sending a signal to one parameter rather than to the other. A $\mathsf{Random}$ Boolean could be implemented as $!\,b(tf).(\boldsymbol{\nu} x)\,(\bar{x}\mid(x.\bar{t}+x.\bar{f}))$, in which the selection is performed "at run-time" by the sum ("+"). A selection made by one process may cause *branching* (also commonly called external choice) in another process. Branching is typically implemented with the $\pi$-calculus sum operator, as in $\bar{b}(\boldsymbol{\nu} tf).(t.P+f.Q)$, which executes $P$ if $b$ is $\mathsf{True}$, and $Q$ if $b$ is $\mathsf{False}$. For example, the "$r = a$ $\mathsf{and}$ $b$" logical circuit can be implemented as follows in the $\pi$-calculus (see Section 2.1).

$$A = !\,r(tf).\bar{a}(\boldsymbol{\nu} t'f').(t'.\bar{b}\langle tf\rangle+f'.\bar{f}), \tag{1.1}$$

Upon receiving a request on $r$, it first queries $a$. If it returns true $(t')$ then $r$ returns the same as $b$. If it returns false instead, $r$ itself returns false $(f)$. So, depending on $a$ and $b$'s behaviour, either a signal will be sent on $t$, or one will be sent on $f$ (but never both).

## 1.6 Dependency Analysis

*Dependency Analysis* is a way to specify the behaviour of a process through logical formulæ, and we use the notation

$$(\Delta_l \blacktriangleleft \Delta_e)$$

where $\Delta_l$ and $\Delta_e$ are *behavioural statements*, to mean that the local process behaves like $\Delta_l$, and the environment *must* behave like $\Delta_e$.

We will introduce a labelled transition system on *Typed Processes* (the pair of a process and a type), that is able to model simultaneous evolution of a process ("$\Delta_l$" above) and constraints on the environment ("$\Delta_e$" above), as in the example seen at the end of Section 1.4.

Describing components of an open system is not only about limiting what interactions are permitted, but also about how third-party processes may provide services required by a process to complete a task. This is covered by *dependencies*. The most general form is provided by *adjoin operators*. For instance the English expression "If $\Delta_e$ holds in the environment then $\Delta_l$ holds in the local process" is formally written $\Delta_l \lhd \Delta_e$. The dependency operator $\lhd$ can be seen as an arrow on a graph, similarly to the dependency graphs used by Yoshida, Berger and Honda in [YBH04], or by Acciai and Boreale in [AB08a]. The difference in meaning between "$(\Delta_l \blacktriangleleft \Delta_e)$" and "$(\Delta_l \lhd \Delta_e)$" may seem rather subtle, so we will come back to it (in Section 4) to explain it in more detail.

Dependency statements are put together using the usual $\vee$ and $\wedge$ connectives. Selection or disjunction $\Delta_1 \vee \Delta_2$ holds in a process if its behaviour matches (at least) one of the $\Delta_i$. Conjunction $\Delta_1 \wedge \Delta_2$ holds in a process if both $\Delta_i$ do. Dependency statements connects *resources* ranged over by Greek letters $\alpha$, $\beta$ and $\gamma$. A resource is typically a behavioural property such as activeness or determinism on some channel, although we'll also see resources involving more than one channel (branching activeness, page 63), or no channel at all (process-level properties, page 94). The statement $\top$ always holds and $\bot$ never does[3].

In summary we use expressions of the following form when making statements about a process:

$$\Delta \; ::= \; (\Delta \lhd \Delta) \;\big|\; (\Delta \vee \Delta) \;\big|\; (\Delta \wedge \Delta) \;\big|\; \top \;\big|\; \bot \;\big|\; \gamma \qquad (1.2)$$

We call productions of this grammar *behavioural statements*. Note how the grammar is essentially that of propositional logic statements, the only custom item being *resources* $\gamma$, which greatly simplifies manipulating behavioural statements and understanding their semantics.

*Dependency Analysis* covers ways of constructing such statements and using them to infer properties about process behaviour.

[3]$\top$ could be considered as an abbreviation of $\bot \lhd \bot$. To keep technical work as simple as possible, however, we will instead show how behavioural statements can be reduced to a form where $\lhd$ only appears in statements of the form $\gamma \lhd \varepsilon$ where $\varepsilon$ is a statement not using $\lhd$, an impossible endeavour if $\top$ isn't a primitive.

Note how most behavioural properties are related to the concept of dependency. Consider the forwarder

$$!\, a(x).\bar{b}\langle x\rangle$$

that just forwards every request to $b$.

Processing caused by a request sent to $a$ eventually terminates if and only if processing caused by a request sent to $b$ does. The $a$-server may cause an information-leak about requests if and only if the $b$-server does. Requests sent to $a$ are eventually answered if and only if requests sent to $b$ are eventually answered. Assuming $a$ and $b$ have the same protocol, as an $a$-server this process respects the protocol if the $b$-server does. As a $b$-client this process respects the protocol if the $a$-client does.

## 1.7 Decidability and Generic Type Systems

A *process type* $\Gamma$ is a structure giving a form of contract between a process and its environment, integrating local and environment behavioural statements with *channel types* describing the protocols to be used at channels, and what type of data they can carry. Whether a process actually holds its part of such a contract is formally specified with *semantic definitions.* Validity of a particular process-type pair $(\Gamma; P)$ (called a typed process) is in general undecidable as processes can have infinitely large transition graphs, making it impossible to fully verify the behaviour of a process. The natural numbers and operators on them can actually be encoded in the $\pi$-calculus, which effectively proves undecidability of type correctness.

Instead, we use a *type system*, a set of rules that tell how to construct a behavioural statement out of a process, in a decidable fashion. My type system is *sound* in the sense that all statements it produces are correct with respect to the semantics. It is however not *complete*, in the sense that it will sometimes fail to notice that a process possesses some property, or will sometimes claim that a resource $\alpha$ depends on the environment providing some other resource $\beta$ when in reality it doesn't.

As in illustration of the distinction between semantic correctness and typability, the former being (by definition) complete and the latter decidable, consider a program iterating through all even numbers larger than two and then trying to decompose them into sums of two prime numbers (again, by checking all smaller prime numbers one by one). If the program ever reaches an even number that doesn't possess such a decomposition, it sends a signal on a channel $\bar{s}$. Such a program, although quite long due to the need to encode natural numbers and checking for primes, can rather easily be written in the $\pi$-calculus. The question of whether the $\bar{s}$-signal will eventually be fired is simply asking which of $\bar{s}_{\mathbf{A}}$ ("$\bar{s}$" *will* be fired) and $\bar{s}_{\mathbf{N}}$ ("$\bar{s}$" *won't* be fired) is a correct statement for that program (the program being deterministic, it is easy to show that precisely one of those two types is correct). This question is equivalent to the Goldbach Conjecture and its answer is not known as of today. Passing that $\pi$-calculus program to our type system (or really any type system written today and known to be sound and decidable[4]) would just return a neutral type, i.e. $\bar{s}$ can't be guaranteed to

[4]I said *known to be* so one can't "cheat" and design a type system that returns the right type when it recognises the Goldbach-testing program

ever be fired, and can't be guaranteed to never be fired, either.

*Typed transition systems* (Section 3.7) predict the evolution of processes from a behavioural point of view (what properties are lost and gained by the process as it interacts with the environment), so that process types effectively provide behavioural information for the entire lifetime of a process and not merely in its current state.

The semantic correctness and type system definitions I will propose in this thesis are *generic* in the sense that they do not work with particular properties, but instead are *parametrised* with respectively *immediate correctness predicates* and *elementary rules* that give the essence of desired properties.

Regarding semantics, given a "good state" predicate, properties can be classified into two groups:

- *safety properties* are those that require the process to *constantly* be in a "good" state, e.g. any input and output on the same channel and both ready to fire and must have the same number of parameters, a process declared deterministic must never face a choice, and so on.

- *liveness properties* are those that require a process to *eventually* reach a "good" state, e.g. a channel declared active must eventually become ready to fire.

Similarly, given elementary rules saying what properties is provided by a single step $\overline{a}\langle\tilde{x}\rangle$ or $a(\tilde{y})$ in a process, the type system recognises two kinds of properties:

- *universal* resources are those that must be provided *everywhere* in the process (usually vacuously), i.e. if a universal resource is not provided by $P$ then it won't be provided by $P \mid Q$ either.

- *existential* resources must be provided *somewhere* in the process, i.e. if an existential resource is provided by $P$ then it is also available in $P \mid Q$.

Note how the two classes of semantics differ in a *temporal* sense ("constantly" versus "eventually") and the two classes of elementary rules have the corresponding difference but in a *spatial* sense ("everywhere" versus "somewhere"), and indeed the soundness theorems hold when connecting universal rules to liveness semantics and existential rules to safety semantics. It is easy to see that attempting to use liveness semantics with an universal rule or safety semantics with an existential rule would fail as for instance the idle process $\mathbf{0}$ vacuously enjoys all universal and safety properties, while that process enjoys no existential or liveness property.

Apart from this essential distinction, the generic type system treats all properties identically with no understanding of their semantics, and the *soundness theorems* show that, if elementary rules suitably imply the corresponding "good state" semantic predicates, statements produced by the instantiated type system are correct with respect to the semantics.

Note that universal and existential properties give rise to safety semantics and existential properties give rise to liveness semantics or, more accurately, the soundness theorems only hold when this correspondence between semantics and elementary rules is

Although one can conceive properties that won't fit neatly in either of these categories (we will discuss a few examples in the course of this thesis) we will see that it is sufficient to cover a wide range of behavioural properties, including *responsiveness* (ability to reliably conduct a conversation), *activeness* (ability to sending/receiving on a channel), *isolation* (lack of measurable side effects), *determinism*, *reachability* (also known as dead code elimination), *termination* and *deadlock-freedom*. These cover more than what is necessary to verify encodings.

## 1.8 Proof-Carrying Behavioural Statements

As the operational semantics of processes are usually given by labelled transition systems (see Definition 2.2.3), the behaviour of a process is usually expressed exclusively in terms of transition sequences, but this has a number of shortcomings, specifically because it usually contains more information than required.

For instance transition sequences distinguish between $a|b \xrightarrow{a}\xrightarrow{b} \mathbf{0}$ and $a|b \xrightarrow{b}\xrightarrow{a} \mathbf{0}$, although those two can be considered as essentially equivalent. Secondly, in a complex system containing loosely related components, a particular run may contain transitions irrelevant to the computation being studied, and those transitions could be simply removed without harm.

Finally, some work is needed to *merge* transition sequences. For example the $a|b \xrightarrow{a}\xrightarrow{b} \mathbf{0}$ sequence can be thought of resulting of the merging of $a|b \xrightarrow{a} b$ and $a|b \xrightarrow{b} a$. This last point is important to deal with interference. For instance one may want to show that any transition sequence $P \xrightarrow{\tilde{\mu}} P'$ can be *continued* into $P \xrightarrow{\tilde{\mu}}\xrightarrow{\tilde{\mu}'_0} Q'$ in such a way that one of those transitions is a communication on some channel $s$. This is obtained by constructing a sequence $P \xrightarrow{\tilde{\mu}_0} Q$ with that property, and showing it can always be merged with $P \xrightarrow{\tilde{\mu}} P'$.

To deal with all these issues we introduce *liveness strategies* (Definition 7.1.1) that basically indicate which components of a process communicate with which. We will show how a particular transition sequence can be transformed in a strategy (or set of strategies in case it mingles unrelated computations), and reciprocally how a strategy can be turned into a transition sequence.

While behavioural statements make claims about processes, they provide no way of verifying those claims. Annotating them with strategies we obtain *proof-carrying behavioural statements* (Definition 7.1.7), which record how the various components were obtained and can easily be transformed into process behaviour.

Although I originally designed them as a proof method, liveness strategies turned out to be interesting in their own right as a means to study the behaviour of a process. A characterisation of determinism is trivial to do using liveness strategies, compared to a definition based on transition labels (Section 8.2). As they permit distinguishing processing of a particular request from unrelated computations, they can be used to transform a process-level property such as determinism or termination into corresponding channel-level properties (Section 7.7).

## 1.9 Contributions of this Thesis

In closing, here is a detailed account of the technical contributions brought by this thesis. I'll start from the giants whose shoulders I've been standing on.

The target calculus, the synchronous $\pi$-calculus with mixed sums and replication (Section 2.1), along with its early labelled transition system (Section 2.2), is well-known and has been widely studied in the past, although I applied some rather inconsequential changes for technical convenience. The idea of using types and type systems for verifying process behaviour is also well-explored. Some examples, including *generic* type systems, are covered in Section 9. Multiplicities were explored by Sangiorgi [San99] and others; independent multiplicities for both ports of a channel were mentioned in Accai and Boreale's [AB08a] on Responsiveness. Kobayashi took the concept beyond what is covered in this thesis, with channel *usages* [Kob02b, Kob02a]. A section in [IK01] suggests using channel types with separate components for inputs and outputs for extra expressiveness, rather than inferring one side from the other, much like my channel types (Section 3.3). Dependency analysis, and more specifically dependent resources have been explored for instance by Honda, Berger, and Yoshida [YBH04]. The notation I use for behavioural statements (Section 3.6) uses $\wedge$ and $\vee$ from propositional logic, and the dependency connective $\triangleleft$, also known as "assume-guarantee" is for instance used by Honda. Finally, my liveness definition (Section 5.2) is reminiscent of Game Theory, and I used the word "strategy" to emphasise this fact.

The contributions of this thesis are the following.

1. *Process Types* (Section 3.4). Seeing the interface between a process and its environment whose parameters are the process' free names, a process type is simply a special case of channel type.

2. *Behavioural statements* (Section 3.6) embedded in process and channel types permit describing resource negotiation with more expressiveness than usual approaches.

3. An extensive *type algebra* covers spatial (composition $\odot$, subtraction $\setminus$, and output-composition $\otimes$, in Section 3.9), logical (equivalence $\cong$, weakening $\preceq$, and reduction $\hookrightarrow$, in Sections 3.6 and 3.11), and dynamical aspects (transition $\wr$ in Section 3.11) of process types.

4. *Safety* (Section 4.3) and especially *liveness semantics* (Section 5.2) integrate a *reliability* component, to provide meaningful results that hold with real schedulers.

5. The types and the type systems are equipped to deal with and describe *choice*, making the type system in Chapter 6, to my knowledge, the first static type system characterising liveness in the presence of choice.

6. My work on structural analysis, initially developed as a proof technique for soundness of the type system with respect to liveness, proved useful in translating process-level properties to channel-level ones, on the semantic level, as well as permitting a compact definition of confluence.

7. The types, type algebra and type systems are *generic*:

  - Behavioural statements integrate arbitrary properties (Section 4.1), for which type operators require no more information than whether they are *universal* or *existential*.
  - Semantics are instantiated with *semantic predicates* that specify if a resource is provided at a specific point *in a process* (Section 4.3). Note that this comes in contrast with other generic type systems that require semantics to be specified in an ad hoc specification language [AB08b], or be to predicates on process *types* [IK01].
  - Type systems are instantiated with *elementary rules* that specify the properties of basic process components (Sections 4.4 and 5.3).

8. Regarding the actual properties covered in this thesis, responsiveness, activeness and isolation are novel and determinism is generalised to a channel-level property:
   - *Responsiveness* (Section 4.8) is a property of channels that respect the protocol defined in channel types.
   - *Activeness* (Section 6) generalises receptiveness, both by distinguishing input and output activeness and including *branching* activeness.
   - *Isolation* (Section 8.1) characterises side-effects.
   - *Determinism* has been studied with more or less the same semantics through dedicated type systems (for instance, [Nes96]), but usually as process-level properties (i.e. the type system tells whether the entire process satisfies the given property), as opposed to channel-level analysis (Section 7.7) that indicates which channels enjoy it.
   - Reachability (Section 8.3), termination (Section 8.4) and deadlock-freedom (Section 8.5) have been studied before but I included them to demonstrate the expressiveness of my generic type systems.

At times, I have also come up with concepts or techniques, only to discover later on that they had been independently developed beforehand. When that happened I tried to synchronise my notation and terminology with pre-exiting ones.

# Chapter 2

# Processes and Operational Semantics

## 2.1 Polyadic $\pi$-Calculus, Guarded Sums, Replication

In order to have a concrete theory and type systems we fix the target calculus, specifically to the synchronous polyadic $\pi$-calculus with guarded sums and replication, but most of the material can be painlessly applied to other process calculi. We use the grammar given in Table 2.1, where $\sigma$ (hereafter usually omitted) stands for $x$'s *channel type*, whose definition is given later. Refer to [Par01] for a more detailed tutorial on the $\pi$-calculus.

The process grammar creates a number of limitations on which processes can be written, that have no effect on the expressiveness (Specifically, all process of the fully general $\pi$-calculus is *strongly bisimilar* [SW01] with one produced by Table 2.1), yet make some proofs easier. First, only guards can be replicated, and they can be replicated only once. This does not limit the expressiveness as $\boldsymbol{!}\,\boldsymbol{!}\,P \sim \boldsymbol{!}\,P$, and both $\boldsymbol{!}\,(P \mid Q)$ and $\boldsymbol{!}\,(P + Q)$ are strongly bisimilar to $(\boldsymbol{!}\,P) \mid (\boldsymbol{!}\,Q)$. Note how the latter simplification removes the need for a (Rep-Comm) rule [SW01] in the labelled transition system, that lets one instance of a replicated process communicate with another, as in $\boldsymbol{!}\,(a.T + \bar{a}) \;\rightarrow T \mid \boldsymbol{!}\,(a.T + \bar{a})$ that gets transformed to $\boldsymbol{!}\,a.T \mid \boldsymbol{!}\,\bar{a} \;\rightarrow T \mid \boldsymbol{!}\,a.T \mid \boldsymbol{!}\,\bar{a}$. Another limitation is that the terms of a sum must be guarded, so that for instance $(a|b) + (c|d)$ is not a valid process, but can be replaced by the strongly bisimilar $a.b + b.a + c.d + d.c$.

Before starting, a little vocabulary, as it is used in this thesis: "Channels" and "names" have their usual $\pi$-calculus meaning, a name being the syntactic element. Unless noted otherwise, lower case Latin letters $a$, $b$, $c$, $d$, $r$, $x$, $y$, $z$ are names. Through renaming, it may happen that two initially different names are assigned to the same channel. A *port* of a channel $a$ is either its input ("$a$") or output ("$\bar{a}$") half. The letters $p$ and $q$ range over ports. A tilde $\tilde{}$ over a symbol stands for a (usually ordered) sequence of elements whose individual elements are represented by the same (tilde-less) symbol with numerical indexes. For instance $\tilde{x}$ stands for $x_1, x_2, \ldots, x_n$.

Free names $\mathrm{fn}(P)$ of a process $P$ are defined as usual, binders being $(\boldsymbol{\nu} x)\,P$

Processes: $P \ ::= \ (P|P) \ \big| \ (\boldsymbol{\nu} x : \sigma)\, P \ \big| \ S \ \big| \ \mathbf{0}$
Components of a parallel composition: $S \ ::= \ (S{+}S) \ \big| \ G.P$
Guards: $G \ ::= \ T \ \big| \ \mathbf{!}\, T$
Non-replicated guards: $T \ ::= \ (\boldsymbol{\nu} x : \sigma)\, T \ \big| \ a(\tilde{y}) \ \big| \ \overline{a}\langle\tilde{x}\rangle$

Table 2.1: Process Syntax

(binding $x$ in $P$) and $a(\tilde{y}).P$ (binding $\tilde{y}$ in $P$).

**Definition 2.1.1 (Subject, Objects and Multiplicities of a Guard)**

- *The* subject port *of a guard $G$ is defined with* $\mathsf{sub}(a(\tilde{y})) \stackrel{\text{def}}{=} a$ *and* $\mathsf{sub}((\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\, \overline{a}\langle\tilde{x}\rangle) \stackrel{\text{def}}{=} \bar{a}$
- *The* object names *are* $\mathsf{obj}(a(\tilde{y})) \stackrel{\text{def}}{=} \tilde{y}$ *and* $\mathsf{obj}((\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\, \overline{a}\langle\tilde{x}\rangle) \stackrel{\text{def}}{=} \tilde{x}$
- *The* bound names *are given by* $\mathrm{bn}(a(\tilde{y})) = \tilde{y}$ *and* $\mathrm{bn}((\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\, \overline{a}\langle\tilde{x}\rangle) = \tilde{z}$
- *Finally $G$ has a* multiplicity $\#(G)$ *equal to $\omega$ if it is replicated,* 1 *otherwise.*

Empty object sets and trailing $\mathbf{0}$ are usually omitted, writing $a$ and $\bar{a}$ instead of $a()$ and $\overline{a}\langle\rangle$, and $G$ instead of $G.\mathbf{0}$.

In order to make some examples easier to read we shall sometimes remove unused bindings, reorder components of a parallel composition or drop idle processes. In other words, we identify processes up to structural congruence in the examples (but this relation plays no significant role in the theory itself). Structural congruence is also helpful to give a succinct definition of top-levelness.

**Definition 2.1.2 (Structural Congruence)** Structural Congruence *on processes is the smallest congruence relation $\equiv$ such that:*

- $(\boldsymbol{\nu} x)\, \mathbf{0} \equiv \mathbf{0}$*, and (for $x \notin \mathrm{fn}(Q)$)* $(\boldsymbol{\nu} x)\, (\boldsymbol{\nu} y)\, P \equiv (\boldsymbol{\nu} y)\, (\boldsymbol{\nu} x)\, P$*,* $((\boldsymbol{\nu} x)\, P)|Q \equiv (\boldsymbol{\nu} x)\, (P|Q)$*,*
- $P|\mathbf{0} \equiv P$*,* $P|Q \equiv Q|P$*,* $P|(Q|R) \equiv (P|Q)|R$*,*
- $P{+}Q \equiv Q{+}P$*,* $P{+}(Q{+}R) \equiv (P{+}Q){+}R$*, and*
- $(P =_\alpha Q) \Rightarrow (P \equiv Q)$ *($\alpha$-renaming).*

## 2.2 Operational Semantics

We end this brief coverage of the $\pi$-calculus with its operational semantics. Rather than having a "program counter" and a fixed program like is common in sequential programming, execution of a $\pi$-calculus process $P$ is expressed by a sequence of *transitions*

$$P \xrightarrow{\mu} P'$$

where $\mu$ contains any input or output done from or to the environment, and $P'$ contains what remains to be executed. For instance the process $a.b.\bar{c}$ (that waits for a signal on $a$ followed by a signal on $b$ before sending one on $c$) is executed as follows:

$$a.b.\bar{c} \xrightarrow{a} b.\bar{c} \xrightarrow{b} \bar{c} \xrightarrow{\bar{c}} \mathbf{0}$$

Remember that the first three processes have an implicit **0** at the end, which must be written explicitly in the fourth, as it is no longer prefixed.

**Definition 2.2.1 (Transition Labels)** Transition labels*, ranged over by* $\mu$*, are given by*

$$\mu \ ::= \ \tau \ \ \big| \ \ a(\tilde{x}) \ \ \big| \ \ (\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\,\overline{a}\langle\tilde{x}\rangle \ \textit{where } a \notin \tilde{z} \subseteq \tilde{x}$$

Note that $(\boldsymbol{\nu}b)\,\overline{a}\langle abab\rangle$ and $a(aa)$ are valid transition labels.

**Definition 2.2.2 (Subject and Objects of a Transition)**

- *The* subject port $\mathsf{sub}(\mu)$ *of a transition label* $\mu \neq \tau$ *is* $\mathsf{sub}(a(\tilde{x})) \stackrel{\text{def}}{=} a$ *or* $\mathsf{sub}((\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle) \stackrel{\text{def}}{=} \bar{a}$.
- *The set of a transition's* objects $\mathsf{obj}(\mu)$ *is given by* $\mathsf{obj}(a(\tilde{x})) \stackrel{\text{def}}{=} \tilde{x}$, $\mathsf{obj}((\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle) \stackrel{\text{def}}{=} \tilde{x}$ *and* $\mathsf{obj}(\tau) = \varnothing$.
- Bound names $\mathrm{bn}(\mu)$ *of* $\mu$ *are* $\mathrm{bn}((\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle) = \tilde{z}$, *and* $\mathrm{bn}(\mu) = \varnothing$ *for other cases.*
- *The set* $\mathrm{n}(\mu)$ *of* names in a transition *is defined as* $\mathrm{n}(a(\tilde{)}) = \tilde{x} \cup \{a\}$, $\mathrm{n}((\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle) = \tilde{x} \cup \{a\}$ *and* $\mathrm{n}(\tau) = \varnothing$.

Note that subjects and objects of guards and of transition labels usually coincide, except that some guards like $\mathbf{!}\,a$ can't be transition labels and some transition labels like $a(a)$ can't be guards.

**Definition 2.2.3 (Labelled Transition System)** *Table 2.2 inductively defines a* transition relation *on processes, and a process $P$ is said to* have *or* do a $\mu$-transition *to process $P'$, if $P \xrightarrow{\mu} P'$.*

*Process $P$* reduces *to $P'$, written $P \ \rightarrow P'$, if $P \xrightarrow{\tau} P'$. It* weakly *reduces to $P'$, written $P \ \Rightarrow P'$ if $P \ \rightarrow \cdots \ \rightarrow P'$ for any number (including zero) of reductions.*

*Finally, $P$ has or does a* weak $\mu$-transition *to $P'$, written $P \overset{\mu}{\Longrightarrow} P'$, if $P \ \Rightarrow \xrightarrow{\mu} \Rightarrow P'$.*

The (Open) and (Com) rules together demonstrate how a private communication channel can be established between two components of a process, using *scope extrusion*, a distinguished feature of $\pi$-calculi. The following example (in a $\pi$-calculus extended with numbers) shows how a client (on the left) sends a query to a server (on the right). The scope of the private reply channel $r$ is extruded when the client sends the request, and the $z \notin \mathrm{n}(\mu)$ condition of (New) makes sure the reply can't be intercepted or faked by a third-party:

$$(\boldsymbol{\nu}r)\,\overline{a}\langle r\rangle.r(y).Q \mid \mathbf{!}\,a(x).\overline{x}\langle 2\rangle \ \rightarrow (\boldsymbol{\nu}r)\,(r(y).Q \,|\, \overline{r}\langle 2\rangle) \mid \mathbf{!}\,a(x).\overline{x}\langle 2\rangle \ \rightarrow$$
$$(\boldsymbol{\nu}r)\,(Q\{^2/_y\}) \mid \mathbf{!}\,a(x).\overline{x}\langle 2\rangle$$

$$\frac{-}{\overline{a}\langle\tilde{x}\rangle.P \xrightarrow{\overline{a}\langle\tilde{x}\rangle} P}\ (\textsc{Out}) \qquad \frac{-}{a(\tilde{y}).P \xrightarrow{a(\tilde{x})} P\{\tilde{x}/\tilde{y}\}}\ (\textsc{Inp})$$

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\tilde{y}:\tilde{\theta})\,\overline{a}\langle\tilde{x}\rangle} Q \quad z \in \tilde{x} \setminus (\{a\} \cup \tilde{y})}{(\boldsymbol{\nu}z : \sigma)\, P \xrightarrow{(\boldsymbol{\nu}z:\sigma,\tilde{y}:\tilde{\theta})\,\overline{a}\langle\tilde{x}\rangle} Q}\ (\textsc{Open})$$

$$\frac{P \xrightarrow{\mu} P'}{!\,P \xrightarrow{\mu} P' \,|\, !\,P}\ (\textsc{Rep}) \qquad \frac{P \xrightarrow{\mu} Q \quad z \notin \mathrm{n}(\mu)}{(\boldsymbol{\nu}z : \sigma)\, P \xrightarrow{\mu} (\boldsymbol{\nu}z : \sigma)\, Q}\ (\textsc{New})$$

$$\frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \varnothing}{P \,|\, Q \xrightarrow{\mu} P' \,|\, Q \qquad Q \,|\, P \xrightarrow{\mu} Q \,|\, P'}\ (\textsc{Par})$$

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\tilde{z}:\tilde{\sigma})\,\overline{a}\langle\tilde{x}\rangle} P' \quad Q \xrightarrow{a(\tilde{x})} Q'}{\begin{array}{c} P \,|\, Q \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\,(P' \,|\, Q') \\ Q \,|\, P \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\,(Q' \,|\, P') \end{array}}\ (\textsc{Com})$$

$$\frac{P \xrightarrow{\mu} P'}{P{+}Q \xrightarrow{\mu} P' \quad Q{+}P \xrightarrow{\mu} P'}\ (\textsc{Sum})$$

$$\frac{P =_\alpha P' \quad P' \xrightarrow{\mu} Q' \quad Q' =_\alpha Q}{P \xrightarrow{\mu} Q}\ (\textsc{Cong})$$

Table 2.2: Labelled Transition System

# Chapter 3

# Simple Types

In this section we introduce *channel types* (that describe what protocol must be used on a channel) and *process types* (that describe both the properties of a process and what is expected from its environment).

A first important fact to note when studying processes is that they often work with many classes of channels with different requirements depending on their role in the program.

This is the reason for introducing *channel types*: rather than expressing properties of a process as a whole, we focus on channels, and associate channel types (written as $\sigma$) to names.

Channel types describe how a process may interact at a channel. A process which exhibits the behaviour declared in channel types is said *well-behaved.* We can then introduce *typed processes* that behave correctly as long as they only interact with well-behaved processes.

## 3.1 Parameter types

As names carried over channels can themselves be used as channels, it becomes quickly obvious that a channel type should include the types of its parameters, like Milner's *sorting* types [Mil93].

For instance, consider the process $P = a(x).\bar{x} \,|\, \overline{a}\langle b\rangle.b(y).\bar{y}$, which reduces to $P \xrightarrow{\tau} \bar{b} \,|\, b(y).\bar{y}$. If $a$'s type does not provide its parameter type, $P$ is not in error right away, but exhibits an arity mismatch after the reduction: The parameter-less output $\bar{x}$ just requires $x$ to be parameter-less, $b(y).\bar{y}$ requires $b$ to have one parameter, and $\overline{a}\langle b\rangle$ requires $a$ to carry one parameter.

This suggests the notation $\sigma \;::=\; (\sigma, \sigma, \ldots, \sigma)$ for a channel type (whose recursion ends at parameter-less channels, written $()$). In the above example, the left component requires $a$ to be of type $\sigma_a = (\sigma_x) = (())$, while the right component requires the type $\sigma_a = (\sigma_b) = ((\sigma_y)) = ((()))$ for $a$, making the type mismatch obvious. A formal definition of errors, including arity and parameter type mismatches, is given in Definition 3.12.1.

## 3.2 Multiplicities

Consider the following situation:

A process $A$ sends a value $v$ to a process $B$, which then sends a reference to the same value to process $C$. As explained before, $A$ actually creates a process $[\![ v ]\!]_u$ encoding the value $v$ into channel $u$, and sends the name $u$ to $B$. $B$ then sends the same name $u$ to $C$. Both $B$ and $C$, to decode the value, send a message on $u$, which is then replied to by $A$. Now $C$ has the potential to change the value $v$ as it appears to $B$, by creating another receiver on $u$. Now, if the scheduler is fair, on average one half of the decoding requests sent by $B$ will actually be intercepted by $C$.

A simple way to solve this issue is with the concept of *multiplicities* [KPT99, San99], which, in their most general form, tell for a channel how many times it may appear in input (respectively, output) subject position. For instance, both ports of $a$ appear in $a|(\boldsymbol{\nu} b)\, b.\bar{a}$ (even though one is deadlocked), and $a$'s input is used once and $b$ not used at all in $(\boldsymbol{\nu} cd)\,(\overline{c}\langle a\rangle \mid \overline{d}\langle b\rangle \mid c(x).x \mid d(y).\mathbf{0})$. We also need to distinguish whether an occurrence is replicated (as $a$ in $!\, a(x).\bar{x}$) or not.

The above issue can now be solved simply by declaring that $u$'s input port has precisely one (replicated) occurrence in subject position, rendering $C$ unable to create one more occurrence without being rejected by a type checker.

The encoding scenario involves the following multiplicities:

1. *Uniform* or $\omega$ names such as $u$ in the example have one replicated input and an arbitrary number of outputs, replicated or not.
2. A decoding request is a message of the form $\overline{u}\langle l\rangle$ where $l$ is *linear*, meaning that it must occur exactly once in output (for the $u$-server to send a reply) and exactly once in input (for the request sender to receive the reply).
3. *Plain* names are those that do not have any requirement.

Other cases may occur, as in the internal choice $\bar{a} \mid a.P \mid a.Q$, where the output port must occur exactly once, and the input port at least once.

Rather than constructing a list of such channel classes we choose to define *port multiplicities* (ranged over by $m$), and record multiplicities independently for input and output ports. To cover the cases seen so far we need three multiplicities: 1, $\omega$ and $\star$, standing respectively for "exactly one non-replicated use", "exactly one replicated use" and "no constraint".[1] We will also need (already in the next section) a multiplicity 0 for ports that must not be used at all.

A natural way of writing this information is to put multiplicities as exponent: $\sigma \ ::= \ (\tilde{\sigma})^{m_i, m_o}$, where $m_i$ is the input multiplicities and $m_o$ the output multiplicities. For instance $\big((\tilde{\sigma})^{1,1}\big)^{\omega,\star}$ would be the type for $u$ in the encoding example, where $\tilde{\sigma}$ describes how such a request is replied (and depends both on the particular encoding and the source calculus type of the encoded value).

## 3.3 Local and Remote uses

All examples we have considered so far have been *input-output-alternating*, in that input processes only output on their parameters. A counter-example is a

[1] The "At least one" case is obtained using $\star$ together with "activeness", as shown later.

"server creator" $\mathbf{!}\,a(x).\mathbf{!}\,x(y).Q$ which creates a one parameter server with body $Q$ on all names sent to it. In that example, a type for $a$ would be of the form $((\sigma)^{\omega,\star})^{\omega,\star}$. However exactly the same type would be given in the case where the input on $x$ is provided by the *output* of $a$ (as in $\overline{a}\langle b\rangle.\mathbf{!}\,b(y).Q$), and yet composing these two processes no longer respects the channel type.

Continuing with the calculus encoding example, we can't require the target processes to have input-output-alternation without requiring processes of the source calculus to have that property (which is most of the time an unreasonable assumption).

This example shows that giving up the input-output-alternation property requires adding information to channel types as to how uses of the parameters are divided between the input and output side of the channel. One way of expressing this information is in terms of a local/remote separation, which is useful because it can easily be adapted to express the interface between a process and its environment, as we will see in the next section.

Instead of merely recording a total number of port uses for a channel, we write the local and the remote uses separately. To fix a notation, we write $\alpha/\beta$ to mean $\alpha$ is local and $\beta$ is remote.

For the parameter uses, we take, as a convention, the point of view of the input process. For instance, consider a two linear parameter channel, whose first parameter is alternating and second is not, as in:

$$a(xy).(\bar{x}|y)\,|\,\overline{a}\langle bc\rangle.(b|\bar{c}) \tag{3.1}$$

The first parameter has multiplicities $0,1/1,0$, while the second has $1,0/0,1$.

Note that this issue only applies to *parameter types*, not to (top-level) channel types. We will provide a similar extension for the channel types in the next section, but, for the moment, distinguish parameter types $\sigma$ and channel types $\pi ::= (\tilde{\sigma})^{m_i,m_o}$. This gives the following syntax for parameter types: $\sigma ::= (\tilde{\sigma})^{m_{li},m_{lo}/m_{ri},m_{ro}}$, where $l$ stands for "local" (i.e., channel's input port), $r$ is remote (channel's output port), $i$ is parameter input and $o$ parameter output.

For instance, 3.1 has, as a type for $a$, $(()^{0,1/1,0},()^{1,0/0,1})^{1,1}$ (in order, $a$'s input does zero input on $x$, one output on $x$, one input on $y$ and zero output on $y$, while $a$'s output does one input on $b$, zero output on $b$, zero input on $c$ and one output on $c$). The outer $1,1$ exponent means that $a$ is a linear name, i.e. is used once in input and once in output.

Note that, even though in this example the parameter multiplicities look very symmetric they need not be so. For instance the type $(()^{0,\star/\star,\star})^{1,1}$ is for a channel whose input side may only use the parameter in output position, but whose output may use the parameter without restrictions.

## 3.4 Process Types

In this section we propose a way to use the channel type notation to describe entire processes, with *process types.*

To explain the similarity between channel and process types we consider the interface between a process $P$ and its environment as a special kind of channel whose parameters are the names free in the process. For instance if $\tilde{z} = \mathrm{fn}(P)$ and $a$ is a fresh name, then $P$'s process type is $a$'s channel type in $a(\tilde{z}).P$. $E$

being a process representing the environment, interaction between $P$ and $E$ may then be modelled as $\tau$-reductions following $\overline{a}(\boldsymbol{\nu}\tilde{z}).E \mid a(\tilde{z}).P \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{z})\,(E \mid P)$.

Using the notation introduced previously, we get $\Gamma ::= (z_1 : \sigma_1, z_2 : \sigma_2, \ldots, z_n : \sigma_n)^{1,1}$ as a notation for a process type, where $z_i$ covers $\tilde{z} = \mathrm{fn}(P)$.

Two things can be noted in that expression. The first is that the exponent $1,1$ is rather uninteresting (it just means "there is one process and there is one environment"). The second is that the $\sigma_i$ are of the form $(\tilde{\sigma})^{m_{li},m_{lo}/m_{ri},m_{ro}}$ rather than $(\tilde{\sigma})^{m_i,m_o}$, i.e. they are parameter types rather than channel types. Note that the local/remote terms make sense now, as these multiplicities tell how the channel usages are divided between local ($P$) and remote ($E$).

Consider for example the process $P = \mathbf{!}\,a(x).\bar{x}$. Wrapping it into an input as described above gives $b(a).P$. In that process, the channel type for $b$ (and therefore the process type for $P$) is $(a : (()^{0,1/1,0})^{\omega,0/0,\star})^{1,1}$. (The "$a$ :" label is used because channel names are not numbered and ordered like channel parameters, but it remains essentially the same as a parameter type.) Now consider a process $E = \overline{a}\langle t\rangle.t$ acting as the environment for $P$. The interaction $P \xrightarrow{a(t)} P|\bar{t} \xrightarrow{\bar{t}} P$ with that process corresponds to the reduction $b(a).P \mid \bar{b}(\boldsymbol{\nu}a).E \rightarrow (\boldsymbol{\nu}a)\big((\mathbf{!}\,a(x).\bar{x}) \mid (\overline{a}\langle t\rangle.t)\big) \rightarrow (\boldsymbol{\nu}a)\big((P|\bar{t}) \mid t\big) \rightarrow (\boldsymbol{\nu}a)\big(P \mid \mathbf{0}\big)$. The process type for the intermediary form $P|\bar{t}$ would be $(a : \sigma_a, t : ()^{0,1/1,0})^{1,1}$, where $\sigma_a$ is $a$'s type seen before. Finally, after $t$ has been consumed, we get $(a : \sigma_a, t : ()^{0,0/0,0})^{1,1}$, expressing the fact that $t$ has been fully used. If, for completeness, we wanted to mention $t$ in the type for $P$ before the first transition, it would have been $t : ()^{0,0/1,1}$, expressing the fact that it may not be used in any way by the process, and the environment may use both ports exactly once. The first transition ($(0,0/1,1) \rightarrow (0,1/1,0)$ on $p$'s multiplicities) can now be seen as $E$ passing $t$'s output capability to $P$.

## 3.5 Types as Triples

In this section we propose a change in channel type notation, to make it more natural and more extensible.

Although there is no serious problem in having channel types of that form in a process type, the issue is that, as the examples showed, multiplicities are not preserved by transitions, while the intuition would suggest that for a channel there should exist a single channel type which remains valid over time.

For instance, in $a|b \xrightarrow{a} b$, $a$'s type ($a$ being assumed linear) evolves as $()^{1,0/0,1} \rightarrow ()^{0,0/0,0}$, and in $a|\bar{a} \xrightarrow{\tau} \mathbf{0}$, $a$'s type evolves as $()^{1,1/0,0} \rightarrow ()^{0,0/0,0}$.

Another issue, perhaps more serious, is that multiplicities are not preserved by composition. For instance, in $P = a|b|\bar{a}$, the first component has $()^{1,0/0,1}$ as a type for $a$, the second has $()^{0,0/1,1}$, the last has $()^{0,1/1,0}$, and in $P$, $a$ has type $()^{1,1/0,0}$. So, in a single process, a single channel has four different types (plus $()^{0,0/0,0}$ which is $a$'s type after the reduction on $a$).

Lastly, the notation introduced here, unlike the one used until now, is easily adaptable to the concept of "parameter protocols" which will be explained later.

All these considerations suggest separating channel types and channel multiplicities, while still keeping the same amount of information.

For a process type we use the notation $(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$, where $\Sigma$ maps names to channel types, $\Xi_{\mathrm{L}}$ contains the local channel usage information and $\Xi_{\mathrm{E}}$ contains the remote channel usage information. Similarly, for channel types we use the notation $(\tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}})$ where $\tilde{\sigma}$ is a set of channel types for the parameters and $\xi_{\mathrm{I}}$, respectively $\xi_{\mathrm{O}}$, gives the parameter multiplicities found in the channel input, respectively output. Note that it is now no longer necessary to distinguish between channel and parameter types.

A channel type $(\tilde{\sigma})^{m_{li},m_{lo}/m_{ri},m_{ro}}$ for a channel $a$ is separated into $(\tilde{\sigma})$, $a^{m_{li},m_{lo}}$ and $a^{m_{ri},m_{ro}}$, and each parameter type $\sigma_i \in \tilde{\sigma}$ is similarly split into its own parameter sequence, input and output behaviour. $i^{m_i,m_o}$ means that parameter number $i$ is used $m_i$ times in input position and $m_o$ times in output position.

For instance, all names being assumed linear, $\overline{a}\langle b\rangle.\bar{b}$ has as a process type $\Gamma = \big(a : \sigma_a, b : (); a^{0,1}, b^{1,1} \blacktriangleleft a^{1,0}, b^{0,0}\big)$: both $a$ and $b$ are locally output once, $b$ is locally input once (as a consequence of being sent to $a$) and $a$ is remotely input once, with $\sigma_a = ((); 1^{1,0}; 1^{0,1})$ (the first parameter is parameter-less, and $a$'s input performs one input on it while $a$'s output performs one output on it).

In the notation used before this section, the same process type would have been written as $(a : (()^{1,0/0,1})^{0,1/1,0}, b : ()^{1,1/0,0})$, omitting the $1,1$ process type exponent. For more clarity we will typically write $a^m, \bar{a}^{m'}$ instead of $a^{m,m'}$. In channel types, terms with zero exponent (such as $1^0$) are usually omitted and so are exponents equal to one (writing for instance $\bar{a}$ instead of $\bar{a}^1$).

In process types, local terms with exponent zero and remote terms with exponent $\star$ are omitted, so that, for instance, the channel $a$ need not be mentioned in a type for process $\mathbf{0}$, as it has local multiplicity zero (in both ports) and remote multiplicity $\star$ for both ports, expressing the fact that the environment has, by default, no constraints on the way it may use the channel.

In that simpler notation, the same process type $\Gamma$ may be written

$$\big(a : \sigma_a, b : (); \bar{a}, b, \bar{b} \blacktriangleleft a, \bar{a}^0, b^0, \bar{b}^0\big)$$

(the process does an output on $a$, an input on $b$ and an output on $b$, while the environment an input on $a$, no output on $a$ and no interaction on $b$), with $\sigma_a = ((); 1; \bar{1})$ (the channel carries a parameter-less channel, its input does an input on the parameter and its output does an output on the parameter).

It should be clear that this new notation, although more extensible and more sound, is precisely as expressive as the previous one, in that any type can be translated from the old to the new notation and *vice versa.* Also note that the representation of a process type as the channel type of an imaginary process-environment communication channel still holds — we use different symbols to emphasise the fact that process types use channel names while channel types use parameter numbers.

## 3.6 Behavioural Statements

The grammar for behavioural statements $\Delta$ is given in (1.2) in the introduction, page 6. Intuitively, *selection* $\Delta_1 \vee \Delta_2$ is correct if one of the $\Delta_i$ does. *Conjunction* $\Delta_1 \wedge \Delta_2$ if both $\Delta_i$ do. $\top$ always holds and $\bot$ never does. The formal semantics rely on several operators and relations on types and will be given later, and further refined as we enrich the algebra (Definitions 3.12.1, 4.3.4 and 5.2.6).

Much like structural congruence on processes we define an equivalence relation $\cong$ on behavioural statements.

**Definition 3.6.1 (Weakening Relation)** *Relation $\preceq$ is the smallest preorder defined by the following rules, where $\cong$ is its symmetric closure. When $\eta_1 \succeq \eta_2$ we say $\eta_1$ is* weaker *than $\eta_2$, and $\eta_2$* stronger *than $\eta_1$. If $\eta_1 \cong \eta_2$, we say $\eta_1$ are* $\cong$-equivalent *or just* equivalent.

1. *On dependencies, behavioural statements or process types (ranged over by $\eta$):*
   - $\eta_1 \wedge \eta_2 \preceq \eta_1 \preceq \eta_1 \vee \eta_2$, *and* $\bot \preceq \eta \preceq \top$.
   - $\eta \wedge (\eta_1 \vee \eta_2) \cong (\eta \wedge \eta_1) \vee (\eta \wedge \eta_2)$ *and* $\eta \vee (\eta_1 \wedge \eta_2) \cong (\eta \vee \eta_1) \wedge (\eta \vee \eta_2)$
   - $\wedge$ *and* $\vee$ *are commutative, associative and idempotent, up to* $\cong$.
   - *If* $\eta_1 \preceq \eta_2$ *then* $\eta \wedge \eta_1 \preceq \eta \wedge \eta_2$ *and* $\eta \vee \eta_1 \preceq \eta \vee \eta_2$.
   - *The $\cong$ relation is a congruence, and $\succeq$ is covariant with respect to $\vee$ and $\wedge$.*
2. *On multiplicities, $m_1 \preceq m_2$ and $p^{m_1} \preceq p^{m_2}$ if $m_1 = 0$ or $m_2 \in \{m_1, \star\}$. Also, $p^\star \cong \top$.*

Some more properties of equivalence and weakening can be derived from the above rules:

**Lemma 3.6.2 (Properties of $\cong$)**

- *Up to $\cong$, $\bot$ is neutral for $\vee$ and absorbent for $\wedge$. $\top$ is absorbent for $\vee$ and neutral for $\wedge$.*
- *Let $C[\cdot]$ and $C'[\cdot]$ be two behavioural contexts and $\varepsilon$ a behavioural statement. Then*
$$C[C'[C[\varepsilon]]] \cong C[C'[\varepsilon]]$$
- *Let $\Delta = \Delta_1 \wedge \Delta_2$, and $\Delta' \succeq \Delta$. Then $\Delta' \cong \Delta'_1 \wedge \Delta'_2$ with $\Delta'_i \succeq \Delta_i$ for both $i$. The same property holds for $\vee$ instead of $\wedge$ or $\preceq$ instead of $\succeq$.*

The proofs are given in Section A.1.1.

The intuitive meaning of equivalence and weakening is that weaker types are correct for more processes, and equivalent types are correct for precisely the same processes. This will be stated formally and proved after we introduce precise definitions of correctness.

As exhaustively describing processes can become rather verbose, we use the following simplifying convention:

**Convention 3.6.3 (Notation for Behavioural Statements)**

1. *In channel types, and in the local component of process types, any port whose multiplicity is not specified is assumed to have multiplicity* 0.
2. *In addition, the local component $\Xi_{\mathrm{L}}$ of a process type with channel type mapping $\Sigma$ should be understood as follows:*

$$\Xi_{\mathrm{L}} \wedge \bigwedge_{x \notin \mathrm{dom}(\Sigma)} \left(x^0 \wedge \bar{x}^0\right)$$

The goal of statements like $p^\star$ ("$p$ is used no more than an infinite number of times), logically equivalent to $\top$, is actually just to prevent the above convention from applying.

Many operators commute with the logical connectives, so, to keep their technical definitions short we introduce:

**Definition 3.6.4 (Logical Homomorphisms)** *A* logical homomorphism *is a function $f$ on behavioural statements or process types is such that $f(X \vee Y) = f(X) \vee f(Y)$ and $f(X \wedge Y) = f(X) \wedge f(Y)$, where, having $\Gamma_i = (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i})$,*

$$\Gamma_1 \ \vee\ \Gamma_2 \overset{\text{def}}{=} (\Sigma_1 \wedge \Sigma_2; \Xi_{\mathrm{L1}} \vee \Xi_{\mathrm{L2}} \blacktriangleleft \Xi_{\mathrm{E1}} \wedge \Xi_{\mathrm{E2}})$$

$$\Gamma_1 \ \wedge\ \Gamma_2 \overset{\text{def}}{=} (\Sigma_1 \wedge \Sigma_2; \Xi_{\mathrm{L1}} \wedge \Xi_{\mathrm{L2}} \blacktriangleleft \Xi_{\mathrm{E1}} \vee \Xi_{\mathrm{E2}})\,.$$

*The $\wedge$ operator on mappings $\Sigma_i$ from names to channel types is equal to their union, provided that the channel types coincide on names they share.*

A logical homomorphism is fully specified by its action on behavioural statements not using $\wedge$ or $\vee$, as the general behaviour can be derived from the above.

Whether two types are related by weakening, $\cong$-equivalent or neither is decidable using a *normal form* for dependency statements but I will defer the proof until Lemma 4.2.13, after introducing behavioural statements including dependencies

The process (1.1) can be given the following type, where the local behavioural statement states that $r$ has multiplicity $\omega$, i.e. has precisely one occurrence which must be replicated. The environment component specifies that $a$ and $b$ must both have at most one replicated instance.

$$\Gamma_A = (a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}; r^\omega \blacktriangleleft a^\omega \wedge b^\omega) \tag{3.2}$$

## 3.7 Typed Transitions

We describe in this section a *transition operator* on types, to answer the following question: If a process $P$ has type $\Gamma$, and $P \xrightarrow{\mu} P'$, what is the type of $P'$? The transition operator applies the transition label $\mu$ to $\Gamma$ and returns $\Gamma \wr \mu$ as a type for $P'$. The motivation is three-fold:

Ruling out transitions that a well-behaved third party process can't cause and that force a process to misbehave. Examples of such illegal transitions are interference with a communication on a linear channel ($l$ being linear, the $l|\bar{l} \xrightarrow{l} \bar{l}$ transition is ruled out, as it contradicts $\bar{l}^0$ in the environment), or ones causing collisions of names of incompatible types. For instance the transition

$$a(x).\overline{x}\langle 3\rangle \mid b(y,z).Q \xrightarrow{a(b)} \overline{b}\langle 3\rangle \mid b(y,z).Q \tag{3.3}$$

introduces an arity mismatch, and is ruled out, as $a$'s parameter's type is incompatible with that of $b$.

Secondly, when using resources and dependencies (Chapters 4 and following), to avoid semantics with universal quantification on third-party processes we characterise the $\triangleleft$ connective with labelled transitions, rather than parallel composition with arbitrary processes (much like barbed congruence is often characterised using labelled bisimilarity). However, labelled transitions change the properties of processes: assume $P$ and $E$ represent a process and its environment. A request $P \xrightarrow{\overline{a}\langle b\rangle}$ is then received as $E \xrightarrow{a(b)} E'$, and if $a$ was assumed *responsive* (Section 4.8) in $E$ then $\bar{b}$ can be assumed to have in $E'$ whatever property is declared in $a$'s channel type. This is implemented again through the transition operator which simultaneously predicts the evolution of the process doing the transition, and of its environment.

Thirdly, in order to prove that the previous point is sound, our "subject reduction" theorem works with arbitrary labelled-transitions (see Proposition 5.6.3 on page 60).

We defer the last two goals for Sections 4 and 5 where we'll formally study behavioural statements involving dependency statements with universal and existential resources.

We start with a definition for transitions without parameters:

**Definition 3.7.1 ($p$-Reduction)** *Let $\Gamma$ be a process type and $p$ a port. Then the $\Gamma \wr p$ operation is the logical homomorphism such that:*

- $p^m \wr p = \begin{cases} \bot & \text{if } m = 0 \\ p^0 & \text{if } m = 1 \\ p^m & \text{if } m \in \{\omega, \star\} \end{cases}$
- *When no other rule applies, $\Delta \wr p = \Delta$.*
- *If $\Gamma = (\Sigma; \Xi_1 \blacktriangleleft \Xi_2)$ then $\Gamma \wr p \overset{\text{def}}{=} (\Sigma; \Xi_1 \wr p \blacktriangleleft \Xi_2 \wr \bar{p})$ (if either of those two operations give $\bot$ then we say instead that $\Gamma \wr p$ is not well-defined).*

On dependency networks, $\Xi \wr p$ is similar but not quite the same as subtraction $\Xi \setminus p^1$ (Definition 3.9.1). The former simulates a transition, and in particular $p^\omega \wr p = p^\omega$ matches $\mathbf{!}\,p \xrightarrow{p} \mathbf{!}\,p$. The latter attempts to "cancel" an application of the $\odot$ operator, and in particular $p^\omega \setminus p$ is *undefined* because the $\Xi \odot p = p^\omega$ equation has no solution (remember that $\mathbf{!}\,p \not\equiv p \,|\, \mathbf{!}\,p$ as the right hand side has type $p^\star$).

An application of the transition operator is not well-defined when it corresponds to an action that no well-typed third-party process would be able to do:

**Definition 3.7.2 (Observability)** *A sum $s = \sum_{i \in I} p_i$ is* observable *in a process type $\Gamma$ (written $\Gamma {\downarrow_s}$) if, for all $i \in I$, $\Gamma \wr p_i$ is well-defined.*

Note how $p^0 \wr p$ produces the neutral element $\bot$ of selection ($\varepsilon \vee \bot \cong \varepsilon$) rather than failing. This is used to prune impossible elements in a selection, when information about the process state gets revealed by transition labels. For instance assume the type $\Gamma$ of a process $P$ has $(a \wedge b) \vee (a^0 \wedge c \wedge d)$ in the local side. Then, if the process follows the transition $P \xrightarrow{a}$, one can safely conclude that the second term of the disjunction is no longer a correct description of the process. And indeed, $\big((a \wedge b) \vee (a^0 \wedge c \wedge d)\big) \wr a = (a \wedge b) \wr a \vee (a^0 \wedge c \wedge d) \wr a = (a^0 \wedge b) \vee (\bot \wedge c \wedge d) = b \vee \bot = b$. This "selection-pruning" becomes very interesting in presence of sums in processes because it precisely mirrors $Q$'s disappearance in the (Sum) rule of the labelled transition system (Table 2.2).

As the definition is symmetric, all these properties apply unchanged for the environment side of a process type.

As an illustration we show on (3.2) how querying a replicated server has no effect on its availability:

$$\begin{aligned}(\Sigma; r^\omega \blacktriangleleft \bar{r}^\star \wedge a^\omega \wedge b^\omega) \wr r &= (\Sigma; r^\omega \wr r \blacktriangleleft (\bar{r}^\star \wedge a^\omega \wedge b^\omega) \wr \bar{r}) \\ &= (\Sigma; r^\omega \blacktriangleleft \bar{r}^\star \wedge a^\omega \wedge b^\omega)\end{aligned}$$

based on $r^\omega \wr r = r^\omega$ and $\bar{r}^\star \wr \bar{r} = \bar{r}^\star$, from the definition.

The full definition of $\Gamma \wr \mu$ (Definition 3.11.2) requires a few more technical elements, that we cover now.

## 3.8 Behavioural Statement Composition

As a counterpart to process parallel composition, we introduce the *process type composition* operator, that answers the following question: If $\Gamma_1$ and $\Gamma_2$ are the types of two processes $P_1$ and $P_2$, what is the type of $P_1|P_2$? This operator, written $\odot$, is of course used by the type system when analysing processes using the parallel composition constructor, but also by the transition operator. The reason is most obvious in presence of replicated inputs: $P = {!}\,a(x).Q \xrightarrow{a(b)} P|(Q\{^b/_x\})$ is mirrored ($P$'s type being $\Gamma$) as $\Gamma \wr a(b) = \Gamma \odot \sigma[b]$, where $\sigma[b]$ injects $b$ into $a$'s channel type $\sigma$ to obtain a type for $Q$.

A port $p$ having multiplicities $m_1$ and $m_2$ in respectively $P_1$ and $P_2$ has multiplicity $m_1 + m_2$ in $P_1 \,|\, P_2$:

**Definition 3.8.1 (Multiplicity Addition)** Multiplicity addition $m+m'$*, has* $0$ *as a neutral element, and returns* $\star$ *for any pair of non-zero multiplicities.*

We first define composition on behavioural statements, before lifting it to full process types.

**Definition 3.8.2 (Behavioural Statement Composition)** Composition *of behavioural statement is done by the logical homomorphism* $\odot$ *such that:*

1. $(p^m) \odot (p^{m'}) \overset{\text{def}}{=} p^{m+m'}$
2. $\Xi \odot \bot \overset{\text{def}}{=} \bot$
3. *When no other rule applies,* $\Delta \odot \Delta' \overset{\text{def}}{=} \top$.

Logical homomorphisms were only defined for single parameter functions but can be generalised to many valued functions using "currification", i.e. seeing $\odot$ as a function mapping behavioural statements to functions mapping behavioural statements to behavioural statements and reading $\Delta_1 \odot \Delta_2$ as $\big(\odot(\Delta_1)\big)(\Delta_2)$. For instance $(\Delta_1 \wedge \Delta_2) \odot (\Delta_3 \vee \Delta_4) = \big((\Delta_1 \odot \Delta_3) \wedge (\Delta_2 \odot \Delta_3)\big) \vee \big((\Delta_1 \odot \Delta_4) \wedge (\Delta_2 \odot \Delta_4)\big)$.

## 3.9 Process Type Composition

When composing process types, the local component "grows" and the environment component "shrinks". Just like the former is described using behavioural statement composition, the latter is described using behavioural statement *subtraction.*

**Definition 3.9.1 (Behavioural Statement Subtraction)** *The* subtraction *operator "$\setminus$" for behavioural statements is defined as follows:*

1. $(p^m) \setminus (p^{m'}) \stackrel{\text{def}}{=} p^{m-m'}$
2. $(\Xi_1 \wedge \Xi_2) \setminus \Xi \stackrel{\text{def}}{=} (\Xi_1 \setminus \Xi) \wedge (\Xi_2 \setminus \Xi)$ *and* $\Xi \setminus (\Xi_1 \wedge \Xi_2) \stackrel{\text{def}}{=} (\Xi \setminus \Xi_1) \wedge (\Xi \setminus \Xi_2)$.
3. *for a set of* $\Xi_i$ *and* $\Xi'_j$ *not using the* $\vee$ *connective:*
$$\bigvee_{i \in I} \Xi_i \setminus \bigvee_{j \in J} \Xi'_j \stackrel{\text{def}}{=} \bigvee_{\rho : J \to I} \bigwedge_{j \in J} (\Xi_{\rho(j)} \setminus \Xi'_j)$$
4. *when no other rules apply,* $\Xi \setminus \Xi' = \Xi$.

Note that unlike composition, subtraction is not commutative or associative, and it is not a logical homomorphism either. In the last point, $\rho$ ranges over all functions with domain $J$ — they do not need to be surjective or injective. We sometimes write $\frac{\Xi}{\Xi'}$ instead of $\Xi \setminus \Xi'$, with the same meaning.

Subtraction and composition of behavioural statements are connected by the following property:

**Lemma 3.9.2 (Subtraction Properties)** *For any three statements* $\Delta_1$, $\Delta_2$ *and* $\Delta_3$*:*
$$\Delta_1 \setminus (\Delta_2 \odot \Delta_2) \cong (\Delta_1 \setminus \Delta_2) \setminus \Delta_3$$

The proof is given in Section A.1.3, as part of the proof of Lemma 3.9.4 below.

We will now describe composition of full process types. This operation builds upon two intuitions:

1. The *local* component of the whole is the composition of the local components of the parts.
2. The *environment* of the whole is the environment of one part, without the local component of the other part.

Formally:

**Definition 3.9.3 (Process Type Composition)** *The* process type composition operator $\odot$ *is defined as follows:*

*Let* $\Gamma_i = (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i})$ *with* $i = 1, 2$. *Then*
$$\Gamma_1 \odot \Gamma_2 \stackrel{\text{def}}{=} \left( \Sigma_1 \wedge \Sigma_2 \, ; \, \Xi_{\mathrm{L1}} \odot \Xi_{\mathrm{L2}} \blacktriangleleft \frac{\Xi_{\mathrm{E1}}}{\Xi_{\mathrm{L2}}} \wedge \frac{\Xi_{\mathrm{E2}}}{\Xi_{\mathrm{L1}}} \right)$$

For instance, for the composition $(x|y) \mid (\bar{y}|\bar{z})$, all channels being linear:

$$\begin{aligned}
&\big(x : (), y : () \quad ; \quad x^1 \wedge y^1 \quad \blacktriangleleft \quad x^0 \wedge \bar{x}^1 \wedge y^0 \wedge \bar{y}^1\big) \odot \\
&\qquad \big(y : (), z : () \quad ; \quad \bar{y}^1 \wedge \bar{z}^1 \quad \blacktriangleleft \quad y^1 \wedge \bar{y}^0 \wedge z^1 \wedge \bar{z}^0\big) = \\
&\qquad\quad \big(\Sigma \quad ; \quad x^1 \wedge y^1 \wedge \bar{y}^1 \wedge \bar{z}^1 \quad \blacktriangleleft \quad x^0 \wedge \bar{x}^1 \wedge y^0 \wedge \bar{y}^0 \wedge z^1 \wedge \bar{z}^0\big),
\end{aligned}$$

where $\Sigma = \{x : (), y : (), z : ()\}$. The local component was obtained by adding zero-terms (Convention 4.2.2):

$$(x^1 \wedge y^1 \wedge \bar{y}^0 \wedge \bar{z}^0) \odot (x^0 \wedge y^0 \wedge \bar{y}^1 \wedge \bar{z}^0) = x^{1+0} \wedge y^{1+0} \wedge \bar{y}^{0+1} \wedge \bar{z}^{0+1},$$

and the environment component is

$$\frac{x^0 \wedge \bar{x}^1 \wedge y^0 \wedge \bar{y}^1}{\bar{y}^1 \wedge \bar{z}^1} \wedge \frac{y^1 \wedge \bar{y}^0 \wedge z^1 \wedge \bar{z}^0}{x^1 \wedge y^1} = \\ (x^0 \wedge \bar{x}^1 \wedge y^0 \wedge \bar{y}^{1-1}) \wedge (y^{1-1} \wedge \bar{y}^0 \wedge z^1 \wedge \bar{z}^0).$$

**Lemma 3.9.4 (Composition Properties)** *The $\odot$ operator is commutative and associative and has element $(\varnothing; \top \blacktriangleleft \top)$ as a neutral element.*

The proof for the general case (types including dependency statements) is given in Section A.1.3. Lemma A.1.5 on page 126 gives more properties of the $\odot$ operator.

I conjecture that, for all semantic definitions used in this thesis, $\Gamma_1$ and $\Gamma_2$ being correct types for respectively $P_1$ and $P_2$ that $\Gamma_1 \odot \Gamma_2$, if well defined, is a correct type for $P_1 \,|\, P_2$. While this seems easy enough to prove for process types without dependency statements (Definition 3.12.1), the proofs becomes difficult for types involving existential resources (Chapter 5). Soundness of the type systems, however, implies that property whenever both $\Gamma_i$ are accepted by the type system for $P_i$.

## 3.10 Channel Instantiation

The channel instantiation operator is used to model the behaviour of an input or output process in reaction to a request, given the type of the channel. It not only sets the channel types of the parameters but also the expected remote behaviour. It acts essentially by substituting parameter references $(1, \ldots, n)$ by the actual parameters $(x_1, \ldots, x_n)$. Extra care is however needed in case two $x_i$ are equal, though, by separating all parameters, doing the substitution and then composing the resulting process types with the $\odot$ composition operator.

Slicing a channel type into parameters is done with a *restriction* operator that is defined much like restriction of a function, and uses the same notation:

**Definition 3.10.1 (Channel Type Restriction)** *The* restriction *of a channel type behavioural statement not using selection "$\vee$", written $\xi|_i$ where $i$ is a parameter number, is a process type inductively defined as follows:*

$$p^m|_i = \begin{cases} p^m & \text{if } \mathrm{n}(p) = i \\ \top & \text{otherwise} \end{cases}$$

$\bot|_i = \bot$, $\top|_i = \top$ *and* $(\xi_1 \wedge \xi_2)|_i = \xi_1|_i \wedge \xi_2|_i$.

*On channel types not using selection,* $(\tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}})\,|_i \overset{\text{def}}{=} (i : \sigma_i; \xi_{\mathrm{I}}|_i \blacktriangleleft \xi_{\mathrm{O}}|_i)$.

*Operators $\wedge$ and $\vee$ are defined for channel types like for process types (Definition 3.6.4), and* $(\sigma_1 \vee \sigma_2)|_i \overset{\text{def}}{=} \sigma_1|_i \vee \sigma_2|_i$ *and* $(\sigma_1 \wedge \sigma_2)|_i \overset{\text{def}}{=} \sigma_1|_i \wedge \sigma_2|_i$ *gives the general case.*

**Definition 3.10.2 (Process Type Complement)** *Let* $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *be a process type. Its* complement $\overline{\Gamma}$ *is then* $(\Sigma; \Xi_{\mathrm{E}} \blacktriangleleft \Xi_{\mathrm{L}})$.

**Definition 3.10.3 (Channel Instantiation)** *Let $\sigma$ be an $n$-ary channel type and $\sigma'$ its completion. Let $\tilde{x}$ be a sequence of $n$ names.*

Input-instantiating *$\sigma$ with $\tilde{x}$ (written $\sigma[\tilde{x}]$) yields the process type*

$$\sigma\big|_1\{^{\tilde{x}}/_{1\ldots n}\} \odot \cdots \odot \sigma\big|_n\{^{\tilde{x}}/_{1\ldots n}\}$$

Output-instantiating $\sigma$ *with* $\tilde{x}$ *(written* $\bar{\sigma}[\tilde{x}]$*) is such that* $\bar{\sigma}[\tilde{x}] = \overline{\sigma[\tilde{x}]}$.

Substitutions apply on entire process types as expected.

Example: Let $\sigma = \big(()(); \bar{1}^1 \wedge 2^1; 1^1 \wedge \bar{2}^1\big)$. Then $\sigma[x,y] = \big(\Sigma; \bar{x}^1 \blacktriangleleft x^1\big) \odot \big(\Sigma; y^1 \blacktriangleleft \bar{y}^1\big) = \big(\Sigma; \bar{x}^1 \wedge y^1 \blacktriangleleft x^1 \wedge \bar{y}^1\big)$ $(\Sigma = \{x : (), y : ()\})$. In that example (and indeed every time all parameters are distinct), $\sigma[x,y]$ is essentially equal to $\sigma\{^{\tilde{x}}/_{1\ldots n}\}$. Performing a $\odot$-composition is necessary if two $x_i$ may be equal: Keeping the same $\sigma$, $\sigma[x,x] = \big(x : (); \bar{x}^1 \blacktriangleleft x^1\big) \odot \big(x : (); x^1 \blacktriangleleft \bar{x}^1\big) = \big(x : (); x^1 \wedge \bar{x}^1; x^0 \wedge \bar{x}^0\big)$. In this case, the input does both the input and the output at $x$, and the output does not interact at it, as told by the $x^0 \wedge \bar{x}^0$ part. For example, $a(xy).(\bar{x} \,|\, y) \mid (\boldsymbol{\nu} b)\,\overline{a}\langle bb\rangle.\mathbf{0} \xrightarrow{\tau} \bar{b} \,|\, b$, where $b$'s linearity is respected.

## 3.11 Transition Operator

To simulate an output transition, one needs to apply the $\odot$ operator but with $\Xi_{\mathrm{L}}$ and $\Xi_{\mathrm{E}}$'s roles exchanged.

**Definition 3.11.1 (Output Composition)** *The* output composition *operator* $\otimes$ *on process types is the binary operator such that* $\overline{\Gamma_1 \otimes \Gamma_2} = \overline{\Gamma_1} \odot \overline{\Gamma_2}$.

Based on process composition and channel type instantiation, we may now generalise Definition 3.7.1:

**Definition 3.11.2 (Transition Operator)** $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *being a process type with* $\Sigma(a) = \sigma$*, the effect of a transition* $\mu$ *on* $\Gamma$ *is* $\Gamma \wr \mu$*, defined as follows.*

- $\Gamma \wr \tau \stackrel{\text{def}}{=} \Gamma$,
- $\Gamma \wr a(\tilde{x}) \stackrel{\text{def}}{=} \Gamma \wr a \odot \sigma[\tilde{x}]$,
- $\Gamma \wr (\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\,\overline{a}\langle\tilde{x}\rangle \stackrel{\text{def}}{=} \Gamma \wr \bar{a} \otimes \bar{\sigma}[\tilde{x}]$.

We conclude this section with the following definitions, that connect transitions on types and transitions on processes, as promised at the beginning of Section 3.7.

**Definition 3.11.3 (Typed Process)** *A* typed process *is a pair* $(\Gamma; P)$ *where* $\Gamma$ *is a process type and* $P$ *a process.*

Note that the above definition imposes no connection whatsoever between the process and the type. When such connection is required I will say it explicitly, for instance $\Gamma$ may have to be semantically correct for $P$ or accepted by the type system.

**Definition 3.11.4 (Transition on Typed Processes)** $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ *if* $P \xrightarrow{\mu} P'$ *and* $\Gamma \wr \mu$ *is well-defined and equal to* $\Gamma'$.

The following Lemma formally states that process types may be considered up to $\cong$ (see also Lemma 3.12.2). We included all operators and relations, even those defined later in this thesis, to avoid fragmenting the lemma.

**Lemma 3.11.5 (Types may be seen up to $\cong$)** *Let $\Delta_1$, $\Delta_2$ be behavioural statements such that $\Delta_1 \cong \Delta_2$.*

*Let $\Phi[\cdot]$ be some expression involving behavioural statements, using only the $\odot$, $\otimes$, $\backslash$, $\wr$, $\mathrm{close}()$ operators and with one "hole" $[\cdot]$. Then $\Phi[\Delta_1] \cong \Phi[\Delta_2]$.*

*Now let $\Phi[\cdot]$ be some statement involving behavioural statements and the above operators, as well as any of the relations $\cong$, $\preceq$, $\searrow$ and $\hookrightarrow$. Then $\Phi[\Delta_1]$ is true iff $\Phi[\Delta_2]$ is.*

## 3.12 Simple Semantics

We have so far given lots of notation and operators, it is now time for *semantic definitions*, to formally tell what is a *correct* type for a process, showing that the algebra makes sense.

The following definition refers to types as described so far as "simple types", as opposed to ones containing dependency statements like $\alpha \lhd \varepsilon$, introduced in the next section.

**Definition 3.12.1 (Simple Semantics)** *Multiplicities and channel types in a typed process $(\Gamma; P)$ are* correct *(written $\Gamma \models_{\#} P$) if, for any sequence $(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'; P')$ with $\Gamma' = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$, the following properties are satisfied:*

1. $\mathrm{dom}(\Sigma) \supseteq \mathrm{fn}(P')$.

2. *If $P' \xrightarrow{\mu} P''$ then there is $\Gamma_+$ and $\mu'$ such that $(\Gamma_+; P') \xrightarrow{\mu'} (\Gamma_+ \backslash \mu'; P'')$ for some $P''$, where $\mu'$ is obtained from $\mu$ by replacing bound objects by fresh names (all distinct in case of inputs), and $\Gamma_+ = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}} \odot \tilde{p}^{\star})$ for some $\tilde{p}$.*

3. *Let $P' \xrightarrow{\mu} P''$ with $p = \mathsf{sub}(\mu)$. If $p^{\omega} \in \Xi_{\mathrm{L}}$ then the derivation for $P \xrightarrow{\mu} P'$ must have used* (Rep) *at some point (i.e. the prefix being consumed in $P$ must be replicated) and $\exists!\ Q$ s.t. $P' \xrightarrow{\mu} Q$ (up to $=_{\alpha}$).*

Point 1 says each free name has a declared type. Point 2 ensures that any transition existing in the process has a corresponding transition in the typed process (which is only possible if the local multiplicities are large enough and if parameter types match). Input objects are replaced by fresh ones to replace transitions like (3.3) by valid ones and some remote multiplicities are replaced by $\star$ to be able to inspect the components of a $\tau$-transitions — for instance we can show that $(\Gamma; P)$ is correct when $\Gamma = \left(l : \lambda; l^1 \wedge \bar{l}^1 \blacktriangleleft l^0 \wedge \bar{l}^0\right)$ and $P = l|\bar{l}$ by setting $\Gamma_+ = \left(l : \lambda; l^1 \wedge \bar{l}^1 \blacktriangleleft l^{\star} \wedge \bar{l}^{\star}\right)$ and checking both $P \xrightarrow{l}$ and $P \xrightarrow{\bar{l}}$. By contrast, $\left(\left(l : \lambda; l^1 \wedge \bar{l}^0 \blacktriangleleft l^0 \wedge \bar{l}^0\right); P\right)$ is not correct because then $\Gamma_+ \wr \bar{l} = \bot$ and thus $P \xrightarrow{\bar{l}}$ has no corresponding transition from $(\Gamma_+; P)$. Note that, for output, this point both proves that (free) output parameters will have types matching the channel type, and that the subject and object's multiplicities are large enough.

Point 3 enforces uniform availability [San99] of $\omega$ names, and prevents $a$ to be marked uniform in $!\,a(x).A \,|\, !\,a(x).B$, because there would be two possible processes resulting from the transition $\mu = a(b)$ rather than one, as required.

From now on we will assume (and, whenever needed, prove!) that multiplicities and channel types in all typed processes being considered are correct.

**Lemma 3.12.2 (Simple Correctness and Structural Equivalence)**
*Let* $\Gamma \models_{\#} P$. *If* $\Gamma \succeq \Gamma'$ *and* $P \equiv P'$ *then* $\Gamma' \models_{\#} P'$ *as well.*

See Section A.1.4 for the proof.

# Chapter 4

# Universal Properties

In this section we introduce behavioural properties (ranged over by $k$), *resources* $p_k$ (ranged over by $\alpha$, $\beta$, $\gamma$ and *dependency statements* "$\gamma \lhd \varepsilon$" into behavioural statements. Intuitively, $\Delta \lhd \Delta'$ holds in a process $P$ if whenever $\Delta'$ holds in $E$, $\Delta$ holds in $P \,|\, E$.

What is the difference between "$\Delta_1 \blacktriangleleft \Delta_2$" and "$\Delta_1 \lhd \Delta_2$"? The former says two things: the process behaves like $\Delta_1$, and the environment is required to satisfy $\Delta_2$. The second statement says that if the environment satisfies $\Delta_2$ then the process will satisfy $\Delta_1$. For instance assume some process $P$ satisfies one of those two statements ($\Delta_1 \blacktriangleleft \Delta_2$ or $\Delta_1 \lhd \Delta_2$). Then composing $P$ with a process $Q$ gives a process $P \,|\, Q$ satisfying $\Delta_1$ if $Q$ satisfies $\Delta_2$. If $Q$ does *not* satisfy $\Delta_2$ then composing $P$ and $Q$ gives a process about which nothing can be said, when the white triangle is used, and fails when the black triangle is used or more specifically the composition of their types with $\odot$ is undefined.

## 4.1 Existential and Universal Resources

A *resource* is an elementary property (such as activeness, isolation, etc) of a channel, a port, or of the process globally.

We represent a resource with the notation $p_k$ where $k$ is a letter representing the property, for instance $p_{\mathbf{F}}$ for functionality or $p_{\mathbf{A}}$ for activeness.

Resources can be classified into two groups depending how they answer the following question:

**Definition 4.1.1 (Universal and Existential Properties)** *If a resource $p_k$ is provided by $P$ but not by $Q$, is $p_k$ available in $P \,|\, Q$? If the answer is yes, $k$ is called an* existential *property, and if the answer is no, $\alpha$ is called a* universal *property. The set of universal properties is written $\mathcal{U}$ and the set of existential properties is written $\mathcal{E}$.*

The names come from analogy with the corresponding quantifiers: A universal (resp., existential) resource $\alpha$ is available in a process $\prod_{i \in I} P_i$ if $\forall i \in I$ (respectively, $\exists i \in I$), $\alpha$ is available in $P_i$.

An example of universal resource is isolation of a channel (every listener must satisfy the isolation requirement), while an example of an existential resource is activeness (e.g. if $P$ eventually sends a message on $\bar{s}$ then this property still holds when third-party processes are added).

As we will see when we move to semantics, existential properties have *liveness* semantics, i.e. they guarantee something ("good") is eventually going to happen, while universal properties have *safety* semantics, i.e. they guarantee that something ("bad") is never going to happen.

The multiplicity statements $p^m$ don't fall neatly in either category, for instance if $P$ provides $p^m$ and $Q$ provides $p^n$ then $P \mid Q$ provides $p^{m+n}$. So multiplicities will typically need special treatment. Dropping the linear multiplicity and only keeping two multiplicities "zero" and "at most finite" (see the discussion on Termination in Section 8.4), in which case they correspond to universal resources.

We now have, in addition to channel type mappings $a : \sigma$, three elementary forms of behavioural statements that can be made about a process: multiplicity statements $p^m$, universal statements $p_k$ with $k \in \mathcal{U}$ and existential statements $p_k$ with $k \in \mathcal{E}$. We will devote the rest of this section in studying process types containing only multiplicity statements (first giving a number of tools for modifying and combining such statements, then providing a precise semantic definition and finally proposing a type system constructing process types from processes.

In this chapter we will focus on universal properties and provide generic semantics and a type system, and reserve treatment of existential properties to Chapter 5 where we will generalise the setting to include existential properties as well.

## 4.2 Universal Type Algebra

We now extend operators seen in the previous section, and introduce a few new ones. The next sections will further extend these operators (in particular to also work on types containing existential resources), so, rather than proving and reproving their properties at every iteration we only prove the most general cases. Most of the time, the specific theorems are merely special cases of the general ones, that appear later in the thesis and are proved in appendices.

The type equivalence relation $\cong$ is extended with the following rules on behavioural statements:

**Definition 4.2.1 ($\triangleleft$-Contravariance)**

$$(\gamma \triangleleft \varepsilon_1) \wedge (\gamma \triangleleft \varepsilon_2) \cong \gamma \triangleleft (\varepsilon_1 \vee \varepsilon_2) \tag{4.1}$$

$$(\gamma \triangleleft \varepsilon_1) \vee (\gamma \triangleleft \varepsilon_2) \cong \gamma \triangleleft (\varepsilon_1 \wedge \varepsilon_2) \tag{4.2}$$

$$\gamma \triangleleft \bot \cong \top \tag{4.3}$$

As with Convention 3.6.3 of page 22 we use a number of notational conventions to keep types concise and readable.

**Convention 4.2.2 (Notation for Behavioural Statements)** *In the rest of this thesis, the following notational conventions apply:*

1. *Priority of operations: $\triangleleft$ binds tighter than $\vee$ and $\wedge$, so the expression $\alpha \wedge \beta \triangleleft \gamma \wedge \delta$ must be read as $\alpha \wedge (\beta \triangleleft \gamma) \wedge \delta$. We will always use brackets in case of ambiguity with respect to $\vee$ or $\wedge$.*

2. *The dependency connective* $\lhd$ *is right-distributive and dependencies can't be nested:*
   - $(\Delta_1 \vee \Delta_2) \lhd \Delta \overset{\text{def}}{=} (\Delta_1 \lhd \Delta) \vee (\Delta_2 \lhd \Delta)$,
   - $(\Delta_1 \wedge \Delta_2) \lhd \Delta \overset{\text{def}}{=} (\Delta_1 \lhd \Delta) \wedge (\Delta_2 \lhd \Delta)$,
   - $(\Delta \lhd \Delta_1) \lhd \Delta_2 \overset{\text{def}}{=} \Delta \lhd (\Delta_1 \wedge \Delta_2)$,
   - $\Delta_1 \lhd (\Delta_2 \lhd \Delta_3) \overset{\text{def}}{=} \Delta_1 \lhd \Delta_2$ *if* $\Delta_3 \ncong \bot$,
   - $\top \lhd \Delta \overset{\text{def}}{=} \top$, *and* $\bot \lhd \Delta \overset{\text{def}}{=} \bot$ *if* $\Delta \ncong \bot$.
   - *Multiplicities* $p^m$ *may not have dependencies, so* $p^m \lhd \varepsilon \overset{\text{def}}{=} p^m$.
3. $p_{k_1 k_2}$ *abbreviates* $(p_{k_1} \wedge p_{k_2})$, *and* $p^m_k$ *means* $p^m \wedge p_k$.
4. *A dependency "*$\lhd \top$*" can be omitted.*
5. *In channel types, and in the local component of process types, any port whose multiplicity and/or universal properties are not specified is assumed to have (respectively) multiplicity* 0 *and/or enjoy all universal properties being considered, without dependencies.*
6. *In addition, the local component* $\Xi_{\mathrm{L}}$ *of a process type with channel type mapping* $\Sigma$ *should be understood as follows:*

$$\Xi_{\mathrm{L}} \wedge \bigwedge_{\substack{x \notin \mathrm{dom}(\Sigma) \\ k \in \mathcal{U}}} \left( x^0 \wedge \bar{x}^0 \wedge x_k \wedge \bar{x}_k \right)$$

Cases with a condition of the form $\Delta \ncong \bot$ are complemented by rule (4.3).

The behavioural statement composition operator $\odot$ is extended with the following rule:

$$(p_k \lhd \varepsilon) \odot (p_k \lhd \varepsilon') \overset{\text{def}}{=} (p_k \lhd \varepsilon) \vee (p_k \lhd \varepsilon') \quad \text{if} \quad k \in \mathcal{U} \tag{4.4}$$

which can also (Definition 4.2.1) be read $(p_k \lhd \varepsilon) \odot (p_k \lhd \varepsilon') \cong p_k \lhd (\varepsilon \wedge \varepsilon')$, capturing the essence of universal properties: Let $P = P_1 \,|\, P_1 \,|\, \ldots \,|\, P_n$, and let, for all $i$, $P_i$'s type (its local component, that is) be $\Xi_i \in \{\gamma,\ \gamma \lhd \bot\}$ for some universal resource $\gamma$. Then, applying (4.4), $P$'s type is $\gamma$ if, *for all* $i$, $\Xi_i = \gamma$. (If $\gamma$ were an existential resource, $P$'s type would be $\gamma$ if *there is* $i$ with $\Xi_i = \gamma$, as we see in the next section on existential resources.)

Type composition may create dependency chains which must then be reduced. For example forwarders

$$a \gg b \overset{\text{def}}{=} \,!\, a(x).\bar{b}\langle x \rangle \tag{4.5}$$

and

$$b \gg c \overset{\text{def}}{=} \,!\, b(x).\bar{c}\langle x \rangle$$

satisfy respectively $a_{\mathbf{R}} \lhd b_{\mathbf{R}}$ (when sending a message to $a$, you will get a response if $b$ is responsive) and $b_{\mathbf{R}} \lhd c_{\mathbf{R}}$ ($\mathbf{R}$ is *responsiveness*, a universal property formally defined in Section 4.8). When composing these two processes as $a \gg b \,|\, b \gg c$, $a_{\mathbf{R}} \lhd b_{\mathbf{R}}$ is still valid, but $a$'s responsiveness depends on $(b_{\mathbf{R}} \wedge c_{\mathbf{R}})$, i.e. $a_{\mathbf{R}} \lhd (b_{\mathbf{R}} \wedge$

$c_{\mathbf{R}}$), because a message sent to $a$ gets resent to $b$ and then ($b$ being free) might either be caught by the $(b \gg c)$-forwarder (in which case we need $c_{\mathbf{R}}$), or it might be caught by another $b$-input in the environment, which is why we also need $b_{\mathbf{R}}$ in order to be guaranteed a reply in all cases.

More generally:

**Definition 4.2.3 (Dependency Reduction)** *The* reduction relation $\hookrightarrow$ *on behavioural statements is a partial order relation satisfying*

1. $(p_k \lhd \varepsilon) \wedge (\gamma \lhd \varepsilon') \hookrightarrow (p_k \lhd \varepsilon) \wedge (\gamma \lhd \varepsilon'\{^{\varepsilon\{^{\perp}/_{\gamma}\} \wedge p_k}/_{p_k}\})$ *for* $k \in \mathcal{U}$.

   *On process types:*
2. $\Xi \hookrightarrow \Xi'$ *implies* $(\Xi \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow (\Xi' \blacktriangleleft \Xi_{\mathrm{E}})$ *and* $(\Xi_{\mathrm{L}} \blacktriangleleft \Xi) \hookrightarrow (\Xi_{\mathrm{L}} \blacktriangleleft \Xi')$.
3. $(\gamma_k \lhd \varepsilon_1 \blacktriangleleft \gamma_k \lhd \varepsilon_2) \hookrightarrow (\gamma_k \lhd (\varepsilon_1 \wedge \varepsilon_2) \blacktriangleleft \gamma_k \lhd \varepsilon_2)$ *for* $k \in \mathcal{U}$.
4. *If* $(\alpha \lhd \varepsilon) \preceq \Xi_{\mathrm{E}}$ *with* $\beta \preceq \varepsilon$ *then* $(\gamma \lhd \varepsilon' \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow$ $\big(\gamma \lhd (\varepsilon'\{^{\alpha \wedge \beta}/_{\alpha}\}) \blacktriangleleft \Xi_{\mathrm{E}}\big)$ *for* $\beta \neq \gamma$.
5. *If* $(\Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow (\Xi'_{\mathrm{L}} \blacktriangleleft \Xi'_{\mathrm{E}})$ *then* $(C[\Xi_{\mathrm{L}}] \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow (C[\Xi'_{\mathrm{L}}] \blacktriangleleft \Xi'_{\mathrm{E}})$ *and* $(\Xi_{\mathrm{L}} \blacktriangleleft C[\Xi_{\mathrm{E}}]) \hookrightarrow (\Xi'_{\mathrm{L}} \blacktriangleleft C[\Xi'_{\mathrm{E}}])$ *for any* local context[1] $C[\cdot]$.

*A behavioural statement* $\Xi$ *is* closed *if* $\Xi \hookrightarrow \Xi'$ *implies* $\Xi \cong \Xi'$. *A* closure *of a behavioural statement* $\Xi$, *written* close $(\Xi)$, *is* $\Xi'$ *such that* $\Xi \hookrightarrow \Xi'$ *and* $\Xi'$ *is closed.*

Point 2 and 5 permit applying reduction on any part of a process type.

Points 3 and 4 permit collapse between the local and environment side of a process type, and is used by the output transition operator $\Gamma \wr \overline{a}\langle\tilde{x}\rangle$ to remove expected remote behaviour from the type. See Section 6.2 for details and an example.

The following Lemma justifies the use of "close" as a function:

**Lemma 4.2.4 (Closure Uniqueness)** *Every behavioural statement has, up to* $\cong$ *(Definition 3.6.1), exactly one closure.*

The proof of the general case (for types including existential properties as well) is given in Appendix A.1.5 on page 129.

Finally, the following definition permits dropping parts of a process types that are no longer used, after a composition:

**Definition 4.2.5 (Removing Non-Observable Dependencies)** *Let* $\Gamma$ *be a process type.* Removing *non-observable dependencies in it is done by the* clean *operator, applying the following operations on its local behavioural statement* $\Xi_{\mathrm{L}}$ *as many times as possible:*

- *Replace any statement* $p_k \lhd \varepsilon$ *where* $p$ *is not observable (Definition 3.7.2) in* $\Gamma$ *by* $\top$
- *In any statement* $\gamma \lhd \varepsilon$, *for any* $p$ *not observable in* $\Gamma$*'s complement* $\overline{\Gamma}$, *replace any* $p_k$ $(k \in \mathcal{U})$ *in* $\varepsilon$ *by* $\top$.

[1] I.e. $C ::= [\cdot] \mid C \wedge \Delta \mid C \vee \Delta$

Process Type composition must now perform dependency reduction, as described in the following updated definition, that

1. first follows Definition 3.9.3,
2. then reduces dependency chains (Definition 4.2.3),
3. finally removes non-observable dependencies (Definition 4.2.5).

**Definition 4.2.6 (Process Type Composition)** Process type composition *applied on two process types* $\Gamma_i = (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i})$ *with* $i = 1, 2$ *(writing* $\Gamma_1 \odot \Gamma_2$*), is equal to:*

$$\mathrm{clean}\left(\mathrm{close}\left(\Sigma_1 \wedge \Sigma_2 \,;\, \Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2} \blacktriangleleft \frac{\Xi_{\mathrm{E}1}}{\Xi_{\mathrm{L}2}} \wedge \frac{\Xi_{\mathrm{E}2}}{\Xi_{\mathrm{L}1}}\right)\right)$$

Note that it is important to first perform dependency reduction and then only remove non-observable non-observable dependencies. For instance when typing $a(x).P \,|\, \overline{a}\langle b\rangle$ where $a$ is linear, the $\bar{a}$-output might require its communication partner to provide some property, i.e. $\gamma \lhd a_k$ for some $\gamma$ and $k \in \mathcal{U}$. Now if $a_k$ depends on $\varepsilon$ in $a(x).P$, the resulting statement becomes $\gamma \lhd (a_k \wedge \varepsilon)$ after closure, and $\gamma \lhd (\top \wedge \varepsilon) \cong \gamma \lhd \varepsilon$ after "cleaning". If the two operations were swapped we'd get $\gamma \lhd \top$, which is incorrect.

Channel instantiation $\sigma[\tilde{x}]$ as used by the transition operator and the type system also needs some adaptation when used with types including dependency statements. The type system treats inputs and outputs by including a model of the remote side, rather than checking the local dependencies are within what's permitted by the protocol. This design reduces protocol violations to circular dependencies. However, we need to make sure that any two parameter resources are related in some way by the protocol:

Consider the process $\overline{a}\langle x, y\rangle$ where $a$ has type

$$(\sigma, \sigma \;;\; 1_{\mathbf{R}} \wedge \bar{2}_{\mathbf{R}} \;;\; \bar{1}_{\mathbf{R}} \wedge 2_{\mathbf{R}})\,. \tag{4.6}$$

for some $\sigma$. From the type we may conclude that both $x_{\mathbf{R}}$ and $\bar{y}_{\mathbf{R}}$ are immediately available (if the input side is active and responsive) and that the process can be considered equivalent to $\overline{a}\langle\ldots\rangle \,|\, x(z).\bar{z} \,|\, \overline{y}(\boldsymbol{\nu} t).t$, and therefore composing it with $y \gg x$ should not create a deadlock. Following the same reasoning, in $a(x, y).P$, $P$ can model the output side as $\overline{x}(\boldsymbol{\nu} z).z \,|\, y(t).\bar{t}$ and therefore setting $P = x \gg y$ should not create a deadlock. Yet of course $\overline{a}\langle x, y\rangle \,|\, y \gg x \,|\, a(x, y).x \gg y$ is deadlocked. This situation arises because each side assumes the remote one will behave according to the channel type.

A simple solution is to *complete* channel types.

**Definition 4.2.7 (Complete Channel Types)** Minimal dependencies *of a remote resource* $\gamma$ *in a behavioural statement* $\xi$ *(written* $\mathrm{md}_\gamma(\xi)$*) are given by the logical homomorphism* $\mathrm{md}_\gamma$ *such that:*

$$\mathrm{md}_\gamma(\alpha \lhd \varepsilon) = \begin{cases} \alpha & \text{if } \gamma \preceq \varepsilon \\ \top & \text{otherwise} \end{cases}$$

Completeness *of a behavioural statement* $\xi$ *with respect to a behavioural statement* $\xi'$ *is defined as follows:*

- $\gamma \lhd \varepsilon$ *is complete with respect to* $\xi'$ *if* $\mathrm{md}_\gamma(\xi_{\mathrm{O}}) \preceq \varepsilon$.
- $\xi_1 \wedge \xi_2$ *is complete with respect to* $\xi'$ *if both* $\xi_i$ *are*
- $\xi_1 \vee \xi_2$ *is complete with respect to* $\xi'$ *if at least one* $\xi_i$ *is.*
- $\top$ *is complete with respect to any* $\xi'$.

**Definition 4.2.8 (Channel Type Completion)** *The* completion $\xi'$ *of a behavioural statement* $\xi$ *is obtained by replacing any statement* $\alpha \lhd \varepsilon$ *in* $\xi$ *by* $\alpha \lhd (\varepsilon \wedge \mathrm{md}_\alpha(\xi'))$.

*Finally, a channel type* $\sigma = (\tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}})$ *is complete if* $\xi_{\mathrm{I}}$ *is complete with respect to* $\xi_{\mathrm{O}}$ *and* $\xi_{\mathrm{O}}$ *is complete with respect to* $\xi_{\mathrm{I}}$. *Let* $\xi'_{\mathrm{I}}$ *be the completion of* $\xi_{\mathrm{I}}$ *for* $\xi_{\mathrm{O}}$, *and* $\xi'_{\mathrm{O}}$ *the completion of* $\xi_{\mathrm{O}}$ *for* $\xi_{\mathrm{I}}$. *Then completing* $\sigma$ *gives* $(\tilde{\sigma}; \xi'_{\mathrm{I}}; \xi'_{\mathrm{O}})$.

**Lemma 4.2.9 (Completion Soundness)** *The completion of a channel type is complete.*

For example, completing (4.6) gives

$$(\sigma, \sigma \quad ; \quad (1_{\mathbf{R}} \wedge \bar{2}_{\mathbf{R}}) \lhd (2_{\mathbf{R}} \wedge \bar{1}_{\mathbf{R}}) \quad ; \quad (\bar{1}_{\mathbf{R}} \wedge 2_{\mathbf{R}}) \lhd (\bar{2}_{\mathbf{R}} \wedge 1_{\mathbf{R}}))$$

which effectively prevents any dependency whatsoever, and would rule out both $\overline{a}\langle x, y\rangle.y \gg x$ and $a(x, y).x \gg y$.

If $a$'s type is instead set to

$$\sigma' = (\sigma, \sigma \;;\; 1_{\mathbf{R}} \wedge \bar{2}_{\mathbf{R}} \lhd \bar{1}_{\mathbf{R}} \;;\; \bar{1}_{\mathbf{R}} \wedge 2_{\mathbf{R}} \lhd 1_{\mathbf{R}}) \tag{4.7}$$

(an instance of a "left-to-right protocol" where parameter resources depend on resources on parameters on their left), the completion is

$$(\sigma, \sigma \;;\; 1_{\mathbf{R}} \lhd \bar{1}_{\mathbf{R}} \;\wedge\; \bar{2}_{\mathbf{R}} \lhd (\bar{1}_{\mathbf{R}} \wedge 2_{\mathbf{R}}) \;;\; \bar{1}_{\mathbf{R}} \lhd 1_{\mathbf{R}} \;\wedge\; 2_{\mathbf{R}} \lhd (1_{\mathbf{R}} \wedge \bar{2}_{\mathbf{R}})) \tag{4.8}$$

for which $\overline{a}\langle x, y\rangle.y \gg x$ is responsive (but $a(x, y).x \gg y$ isn't).

Restriction (Definition 3.10.1) of dependency statements is done as follows:
$\gamma \lhd \varepsilon|_i = \begin{cases} \gamma \lhd \varepsilon & \text{if } \mathrm{n}(\gamma) = i \\ \top & \text{otherwise} \end{cases}$ Even though there is one factor for each parameter, one factor may refer to resources defined in other factors, in case of conditional parameter resources.

The parameter instantiation operator is otherwise unchanged, following Definition 3.10.3, except that if it results in self-dependent statements $\gamma \lhd \varepsilon$ where $\gamma$ appears in $\varepsilon$ then $\varepsilon$ must additionally substitute any $\gamma$ in $\varepsilon$ by $\perp$.

For instance: having $\sigma'$ as in (4.7), $\sigma'[x, y]$ is

$$\begin{aligned} &(x : \sigma \;;\; x_{\mathbf{R}} \lhd \bar{x}_{\mathbf{R}} \;\blacktriangleleft\; \bar{x}_{\mathbf{R}} \lhd x_{\mathbf{R}}) \odot \\ &\qquad\qquad (y : \sigma \;;\; \bar{y}_{\mathbf{R}} \lhd (\bar{x}_{\mathbf{R}} \wedge y_{\mathbf{R}}) \;\blacktriangleleft\; y_{\mathbf{R}} \lhd (x_{\mathbf{R}} \wedge \bar{y}_{\mathbf{R}})) = \\ &(x : \sigma, y : \sigma \;;\; x_{\mathbf{R}} \lhd \bar{x}_{\mathbf{R}} \;\wedge\; \bar{y}_{\mathbf{R}} \lhd (\bar{x}_{\mathbf{R}} \wedge y_{\mathbf{R}}) \;\blacktriangleleft\; \bar{x}_{\mathbf{R}} \lhd x_{\mathbf{R}} \;\wedge\; y_{\mathbf{R}} \lhd (x_{\mathbf{R}} \wedge \bar{y}_{\mathbf{R}})) \end{aligned} \tag{4.9}$$

which is (4.8) but with $x$ and $y$ substituting 1 and 2. As we saw after Definition 3.10.3 for multiplicities, it may once more seem a lot of unnecessary trouble to slice the channel type, substitute and compose it back, if the result is just a substitution of parameter numbers with parameter names. But consider a channel type whose input side contains $1_{\mathbf{R}}\lhd\varepsilon \wedge 2_{\mathbf{R}}\lhd\varepsilon'$. Then (output) parameter instantiation setting both 1 and 2 to the same name $x$ will transform that expression to $x_{\mathbf{R}}\lhd\varepsilon \odot x_{\mathbf{R}}\lhd\varepsilon' = x_{\mathbf{R}}\lhd(\varepsilon\wedge\varepsilon')$ (contrast with $x_{\mathbf{R}}\lhd\varepsilon\wedge x_{\mathbf{R}}\lhd\varepsilon' \cong x_{\mathbf{R}}\lhd(\varepsilon\vee\varepsilon')$ which is what you'd get with substitution).

We now generalise the transition operator (Definition 3.11.2) to types including behavioural statements. The basic idea is that, having $P \xrightarrow{a(\tilde{x})} P'$, $P \,|\, \overline{a}\langle\tilde{x}\rangle \xrightarrow{\tau} P'$ so, as types should be preserved by reduction, $P'$ should have the same type as $P \,|\, \overline{a}\langle\tilde{x}\rangle$. Specifically[2]:

**Definition 4.2.10 (Transition Operator with Universal Properties)** $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *being a process type with* $\Sigma(a) = \sigma$ *and* $a$*'s multiplicities in* $\Gamma$ *being* $m$ *and* $m'$*, the effect of a transition* $\mu$ *on* $\Gamma$ *is* $\Gamma \wr \mu$*, defined as follows.*

- $\Gamma \wr \tau \overset{\mathrm{def}}{=} \Gamma$,
- $\Gamma \wr a(\tilde{x}) \overset{\mathrm{def}}{=} \Gamma \wr a \odot \sigma[\tilde{x}] \odot \mathsf{prop}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle, \sigma, m, m')$,
- $\Gamma \wr (\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\,\overline{a}\langle\tilde{x}\rangle \overset{\mathrm{def}}{=} \Gamma \wr \overline{a} \otimes \overline{\sigma}[\tilde{x}] \odot \mathsf{prop}_{\mathcal{K}}(a(\tilde{x}), \sigma, m, m')$.

Refer to Section 8.1 for an example of this operator with isolation ($\mathcal{K} = \{\mathbf{I}\}$). Note that it reduces to Definition 3.11.2 when $\mathcal{K} = \varnothing$. For some properties (such as responsiveness, see Section 4.8) the extra $\mathsf{prop}_{\mathcal{K}}$-term is actually unnecessary as it only provides properties of the subject, which are irrelevant after it is consumed. In those cases, dropping it would preserve correctness and yield a stronger type. The most common case is actually that $\mathsf{prop}_{\mathcal{K}}$ produces terms that must be preserved after the transitions, and some terms that could be dropped. We will discuss this further when working on process level properties (Section 7.7). A stronger transition operator could therefore be obtained by splitting the $\mathsf{prop}_{\mathcal{K}}$-function in a "subject-related" (that aren't needed by the transition operator) and a "global" term (that must be included by the transition operator), but I won't do so for simplicity.

Deciding if two types are equivalent or related by weakening can be done by constructing their *normal forms.* We use the $\bigvee_{i\in I}\Delta_i$ notation to mean $\Delta_1 \vee \Delta_2 \vee \ldots$, and similar for $\wedge$. $\bigvee_{i\in\varnothing}\Delta_i \overset{\mathrm{def}}{=} \bot$ and $\bigwedge_{i\in\varnothing}\Delta_i \overset{\mathrm{def}}{=} \top$.

**Definition 4.2.11 (Normal Form)** *A behavioural statement or dependency* $\eta$ *is in* normal form *if it satisfies the following properties:*

1. $\eta' = \bigvee_{i\in I}\eta_i$ *and* $\eta_i = \bigwedge_{j\in I_j}\eta_{ij}$ *where* $\eta_{ij}$ *are either resources (for the normal form of a dependency) or dependency statements (for the normal form of a behavioural statement) whose dependencies are themselves in normal form*
2. *The sets* $I$ *and* $I_j$ *are minimal.*

Although statements can have more than one normal form, they are all $\cong$-equivalent to each other, as $\cong$ is an equivalence relation.

[2]We should technically write something like "$\Gamma \wr_{\mathcal{K}} \mu$" as the definition relies on $\mathcal{K}$, but do not do so to keep notation readable.

**Lemma 4.2.12 (Normal Form)** *Any behavioural statement or dependency is $\cong$-equivalent to at least one behavioural statement or dependency in normal form.*

*Proof* In the context of constructing a normal form, two statements *match* if they can be merged in some way, when connected by $\vee$ or $\wedge$. Specifically: Every statement matches itself as for instance $\varepsilon \vee \varepsilon \cong \varepsilon$ by idempotence, two statements $\gamma \lhd \varepsilon_1$ and $\gamma \lhd \varepsilon_2$ match as f.i. $(\gamma \lhd \varepsilon_1) \vee (\gamma \lhd \varepsilon_2) \cong \gamma \lhd (\varepsilon_1 \wedge \varepsilon_2)$.

Two conjunctions $\bigwedge_{i\in I} \Delta_i$ and $\bigwedge_{i'\in I'} \Delta_{i'}$ match if either every $\Delta_i$ with $i \in I$ is $\cong$-equivalent to some $\Delta_{i'}$ for some $i' \in I'$, or if there are $\hat{\imath} \in I$ and $\hat{\imath}' \in I'$ such that $\Delta_{\hat{\imath}}$ matches $\Delta_{\hat{\imath}'}$, every $\Delta_i$ with $i \neq \hat{\imath}$ is $\cong$-equivalent to some $\Delta_{i'}$, and reciprocally every $\Delta_{i'}$ with $i' \neq \hat{\imath}'$ is $\cong$-equivalent to some $\Delta_i$.

In the former case, $\bigwedge_{i\in I} \Delta_i \vee \bigwedge_{i'\in I'} \Delta_{i'} \cong \bigwedge_{i'\in I'} \Delta_{i'}$ (as a consequence of $(\Delta_1 \cong \Delta_2) \Rightarrow \big((\Delta_1 \vee \Delta_2) \cong \Delta_1\big)$). In the latter case, $\bigwedge_{i\in I} \Delta_i \vee \bigwedge_{i'\in I'} \Delta_{i'} \cong \bigwedge_{i\in I\setminus\{\hat{\imath}\}} \Delta_i \wedge (\Delta_{\hat{\imath}} \vee \Delta_{\hat{\imath}'})$ (as a consequence of the $(\Delta \wedge \Delta_1) \vee (\Delta \wedge \Delta_2) \cong \Delta \wedge (\Delta_1 \vee \Delta_2)$ rule).

Note that matching is symmetric and reflexive but *not* transitive. For instance $\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \varepsilon_\beta \wedge \gamma \lhd \varepsilon_\gamma$ matches both $\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \hat{\varepsilon}_\beta \wedge \gamma \lhd \varepsilon_\gamma$ and $\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \varepsilon_\beta \wedge \gamma \lhd \hat{\varepsilon}_\gamma$ but the latter two don't match each other.

The normal form of a behavioural statement is constructed so that in any conjunction or disjunction appearing in it, no two terms match each other, thereby being in some sense "minimal". We show by structural induction that any behavioural statement $\Delta$ has such a normal form.

For $\Delta = \bot$ and $\Delta = \top$ the normal forms are respectively $\bigvee_{i\in\varnothing} \varepsilon_i$ and $\bigwedge_{i\in\varnothing} \varepsilon_i$.

Let $\Delta$ and $\Delta'$ be two behavioural statements with normal forms $\bigvee_{i\in I} \Delta_i$ and $\bigvee_{i'\in I'} \Delta_{i'}$. The normal form of $\Delta \vee \Delta'$ is obtained from $\bigvee_{i\in I\cup I'} \Delta_i$ by merging pairs of matching $\Delta_i$ as indicated above until it is no longer possible. Although the $\Delta_i$ were themselves irreducible, merging them may introduce matching subterms, which can inductively be reduced.

Let $\Delta_{i,i'}$ be the normal form of $\Delta_i \wedge \Delta_{i'}$. The normal form of $\Delta \wedge \Delta'$ is then obtained from $\bigvee_{i\in I,\, i'\in I'} (\Delta_{i,i'})$ by merging pairs of matching $\Delta_{i,i'}$ until no longer possible. Again, the merging may permit further simplification. □

The following two rules can be used to directly verify if two types are related by weakening, given their normal forms:

**Lemma 4.2.13 (Weakening Criteria)** *Let $\{\varepsilon_i\}_{i\in I}$ and $\{\varepsilon_j\}_{j\in J}$ be sets of dependencies. Then:*

1. $\bigvee_{i\in I} \varepsilon_i \preceq \bigvee_{j\in J} \varepsilon_j$ *if for all $j \in J$, there is $i \in I$ such that $\varepsilon_i \preceq \varepsilon_j$.*

2. $\bigwedge_{i\in I} \varepsilon_i \preceq \bigwedge_{j\in J} \varepsilon_j$ *if for all $i \in I$, there is $j \in J$ such that $\varepsilon_i \preceq \varepsilon_j$.*

The proof is given in Section A.1.6.

Note that this Lemma is not complete, as statements may have many normal forms that are not directly comparable with it. For instance $\big(\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \varepsilon_\beta \wedge \gamma \lhd \varepsilon_\gamma\big) \vee \big(\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \hat{\varepsilon}_\beta \wedge \gamma \lhd \varepsilon_\gamma\big) \vee \big(\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \varepsilon_\beta \wedge \gamma \lhd \hat{\varepsilon}_\gamma\big)$ has two normal forms $\big(\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd (\varepsilon_\beta \wedge \hat{\varepsilon}_\beta) \wedge \gamma \lhd \varepsilon_\gamma\big) \vee \big(\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \varepsilon_\beta \wedge \gamma \lhd \hat{\varepsilon}_\gamma\big)$ and $\big(\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \varepsilon_\beta \wedge \gamma \lhd (\varepsilon_\gamma \wedge \hat{\varepsilon}_\gamma)\big) \vee \big(\alpha \lhd \varepsilon_\alpha \wedge \beta \lhd \hat{\varepsilon}_\beta \wedge \gamma \lhd \varepsilon_\gamma\big)$, necessarily $\cong$-equivalent, yet not comparable using Lemma 4.2.13.

## 4.3 Universal Semantics

In this section I outline semantic definitions for universal properties using the notation $\Gamma \models_{\mathcal{U}} P$, read "The universal properties in process type $\Gamma$ form a correct description of process $P$".

Universal properties have what is commonly called *safety* semantics in the literature (Not all safety properties can be expressed as universal properties, however — linearity is a counter-example). For instance consider the universal property $\mathbf{N}$ such that $p_{\mathbf{N}}$ means "$p$ never appears in subject position". It is universal in the sense that, in order to be available in $P \mid Q$, it must hold in both $P$ and $Q$. It is a safety property in the sense that it can be *disproved* by a sequence $P \xrightarrow{\tilde{\mu}} P'$ if $p$ appears in subject position in process $P'$. That property is further explored in Section 8.3.

More generally, semantics of a universal property $k$ is provided by a *semantic predicate*:

**Definition 4.3.1 (Semantic Predicate)** *A* semantic predicate $\mathsf{good}_k$ *is defined for pairs* $(p\triangleleft\varepsilon, (\Gamma; P))$ *and satisfies these conditions:*

- *If* $\varepsilon \succeq \varepsilon'$ *then* $\mathsf{good}_k(p\triangleleft\varepsilon, (\Gamma; P)) \Rightarrow \mathsf{good}_k(p\triangleleft\varepsilon', (\Gamma; P))$.
- *If* $P \equiv P'$ *then* $\mathsf{good}_k(p\triangleleft\varepsilon, (\Gamma; P)) \iff \mathsf{good}_k(p\triangleleft\varepsilon, (\Gamma; P'))$.
- $\mathsf{good}_k(p\triangleleft\varepsilon, (\Gamma; P)) \iff \mathsf{good}_k(p\{^{\tilde{x}}/_{\tilde{y}}\}\triangleleft\varepsilon\{^{\tilde{x}}/_{\tilde{y}}\}, (\Gamma\{^{\tilde{x}}/_{\tilde{y}}\}; P\{^{\tilde{x}}/_{\tilde{y}}\}))$ *for any injective substitution* $\{^{\tilde{x}}/_{\tilde{y}}\}$.

If the predicate value is preserved by strong bisimulation ($P \sim P'$ implies $\mathsf{good}_k(p\triangleleft\varepsilon, (\Gamma; P)) \iff \mathsf{good}_k(p\triangleleft\varepsilon, (\Gamma; P'))$), it is said a *behavioural* property, and even a *weak* behavioural property if it is preserved by weak bisimulation $\approx$.

The value of a semantic predicate should not be related to the correctness of the typed process $(\Gamma; P)$, but it may use information from it (typically, $p$'s multiplicities) to decide if $p_k\triangleleft\varepsilon$ is an error. That dependency statement does not even necessarily appear in $\Gamma$.

A semantic predicate characterises some universal property if it satisfies the following:

**Definition 4.3.2 (Universal Predicate, Error State)** *A predicate* $\mathsf{good}_k$ *is* universal *if, whenever* $\Gamma\odot\Gamma'$ *is well-defined,* $\mathsf{good}_k(p\triangleleft\top, (\Gamma\odot\Gamma'; P \mid P'))$ *implies* $\mathsf{good}_k(p\triangleleft\top, (\Gamma; P))$.

*A typed process* $(p_k\triangleleft\varepsilon; P)$ *with* $k \in \mathcal{U}$ *is said in* error state *if* $\mathsf{good}_k(p\triangleleft\varepsilon, (\Gamma; P))$ *is false.*

We will see the existential counterpart in the next section.

Any behavioural statement $\Xi_2$ can be written as a conjunction of disjunctions, and then any of those disjunctions ($\Xi_0$ in the definition below) is called a projection of $\Xi_2$. Formally:

**Definition 4.3.3 (Elementary Statements, Projections)**
*An* elementary statement *is a behavioural statement of the form*

$$\bigvee_i (\gamma_i\triangleleft \bigwedge_j \alpha_{ij}).$$

*Let $\Xi_2$ be a behavioural statement. An elementary statement $\Xi_0$ is a* projection *of $\Xi_2$, written $\Xi_2 \searrow \Xi_0$, iff. $\Xi_2 \succeq \Xi_0$ and, for all elementary statements $\Xi_1$ such that $\Xi_2 \succeq \Xi_1 \succeq \Xi_0$, we have $\Xi_0 \cong \Xi_1$.*

Given safety predicates for all universal properties, correctness of a typed process is given by the following:

**Definition 4.3.4 (Universal Semantics)** *Let $(\Gamma; P)$ be a typed process. It is said* correct with respect to universal semantics *(written $\Gamma \models_{\mathcal{U}} P$) if, for all transition sequences $(\Gamma; P) \xrightarrow{\tilde{\mu}}\searrow (\Gamma'; P')$, the local component of $\Gamma'$ having a normal form $\bigvee_{i\in I} p_{i k_i} \triangleleft \varepsilon_i$, for all $i \in I$ with $k_i \in \mathcal{U}$, $\mathsf{good}_{k_i}(p_i \triangleleft \varepsilon_i, (\Gamma'; P'))$ holds.*

We will see examples of $\mathsf{good}_k$ in Section 8.

## 4.4 Universal Type System

In this section we will discuss my decidable and sound but of course not complete type system for a set of properties $\mathcal{K} \subseteq \mathcal{U}$.

Just like the semantic correctness is parametrised by a safety predicate $\mathsf{good}_k$, the type system is parametrised by an operator $\mathsf{prop}_k$ giving the local properties satisfied by a guard:

**Definition 4.4.1 (Elementary Guard Rule)** *Let $k$ be a property. The corresponding* elementary guard rule*, denoted $\mathsf{prop}_k$, is a function mapping tuples $(G, \sigma, m, m')$, where $G$ is a guard, $\sigma$ is a (its subject's) channel type, and $m$, $m'$ are multiplicities (total input and output multiplicities of $G$'s subject), to behavioural statements $\Xi$ and is such that, for all $G$, $\sigma$, $m$ and $m'$:*

- *The returned $\Xi$-statements must be $\bot$ or of the form $\bigwedge_{i\in I} \gamma_i \triangleleft \varepsilon_i$, with all $\gamma_i$ being $k$-resources.*
- $q_k \triangleleft \varepsilon \succeq \mathsf{prop}_k(\sigma, G, m, m') \quad \Rightarrow \quad \mathsf{good}_k(q \triangleleft \varepsilon, (\left(p : \sigma;\ \blacktriangleleft p^m \wedge \bar{p}^{m'}\right); G))$ *(taking $p = \mathsf{sub}(G)$).*
- $\mathsf{prop}_k(\sigma, G\{\tilde{x}/\tilde{y}\}, m, m') \cong \mathsf{prop}_k(\sigma, G, m, m')\{\tilde{x}/\tilde{y}\}$

We don't allow elementary rules to produce disjunctions because it makes some proofs simpler, but they wouldn't create any particular difficulties.

$\mathsf{prop}_k(\sigma, G, m, m')$ may use $p$'s multiplicities as well as its object resources' dependencies (even if objects are bound) to compute $p_k$'s multiplicities.

**Definition 4.4.2 (Local Properties)** *A property $k$ is* local *if its elementary rule $\mathsf{prop}_k(\sigma, G, m, m')$ always yields either $\bot$ or statements of the form $\bigwedge_{i\in I} \gamma_i$.*

Similarly, an operator $\mathsf{sum}_k$ giving the local properties satisfied by a sum (the sum itself, not the individual guards — for instance the Activeness instance introduces *sum activeness* and the Determinism instance introduces *process-level non-determinism*).

$$\frac{-}{(\varnothing ; \top \blacktriangleleft \top) \vdash_{\mathcal{K}} \mathbf{0}}\ (\text{U-NIL})$$

$$\frac{\forall i : \Gamma_i \vdash_{\mathcal{K}} P_i}{\Gamma_1 \odot \Gamma_2 \vdash_{\mathcal{K}} P_1 \mid P_2}\ (\text{U-PAR}) \qquad \frac{\Gamma \vdash_{\mathcal{K}} P \qquad \Gamma(x) = \sigma}{(\boldsymbol{\nu} x)\, \Gamma \vdash_{\mathcal{K}} (\boldsymbol{\nu} x : \sigma)\, P}\ (\text{U-RES})$$

$$\frac{\begin{array}{c}\forall i : (\Sigma_i ; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash_{\mathcal{K}} G_i . P_i \\ \Xi_{\mathrm{E}} \preceq \bigwedge_i \Xi_{\mathrm{E}i}\end{array}}{\left(\bigwedge_i \Sigma_i ; \bigwedge_{k \in \mathcal{K}} \mathsf{sum}_k(\{p_i\}_i, \Xi_{\mathrm{E}}) \wedge \bigvee_i \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}}\right) \vdash_{\mathcal{K}} \sum_i G_i . P_i}\ (\text{U-SUM})$$

$$\frac{\Gamma \vdash_{\mathcal{K}} P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x}}{\begin{array}{rl} \left(p : \sigma ;\ \blacktriangleleft p^{m} \wedge \bar{p}^{m'}\right) & \odot \\ (; p^{\#(G)} \blacktriangleleft) & \odot \\ !_{\text{if } \#(G) = \omega}\ (\boldsymbol{\nu} \mathrm{bn}(G)) \Big( \Gamma & \odot \\ (; \bigwedge_{k \in \mathcal{K}} p_k \lhd \mathsf{prop}_k(\sigma, G, m, m') \blacktriangleleft) & \odot \\ \overline{\sigma}[\tilde{x}] \Big) & \vdash_{\mathcal{K}} G.P \end{array}}\ (\text{U-PRE})$$

Table 4.1: Universal Type System Rules

**Definition 4.4.3 (Elementary Sum Rule)** *Let $k$ be a property. The corresponding* elementary sum rule $\mathsf{sum}_k$ *is a function mapping pairs $(\tilde{p}, \Xi)$, where $\tilde{p}$ is an unordered sequence of ports (the sum guards) and $\Xi$ a behavioural statement (the process type's environment component) to behavioural statements $\Xi$ and is such that, for all $\tilde{p}$, $\Xi$:*

- $q_k \lhd \varepsilon \succeq \mathsf{sum}_k(\{p_i\}_i, \Xi_{\mathrm{E}}) \ \Rightarrow\ \mathsf{good}_k(q \lhd \varepsilon, ((\Sigma;\ \blacktriangleleft \Xi_{\mathrm{E}}) ; \sum_i G_i . P_i))$ *if* $\forall i :$ $\mathsf{sub}(G_i) = p_i$
- $\mathsf{sum}_k(\tilde{p}\{^{\tilde{x}}/_{\tilde{y}}\}, \Xi\{^{\tilde{x}}/_{\tilde{y}}\}) \ \cong\ \mathsf{sum}_k(\tilde{p}, \Xi\{^{\tilde{x}}/_{\tilde{y}}\})\{^{\tilde{x}}/_{\tilde{y}}\}$

Given a process $P$, a mapping $\Sigma$ of channel types for all free names, and optionally multiplicities for some names, the type system constructs a process type $\Gamma$ for $P$. Processes deemed unsafe (that may violate multiplicity constraints or mismatch channel types) are rejected as untypable. Incompleteness means that typing may fail for process that are actually safe, and even when typing succeeds, the behavioural statement constructed by the type system may be weaker than what is actually satisfied by the type system.

**Definition 4.4.4 (Universal Type System)** Typability *of a typed process $(\Gamma ; P)$ with respect to a set of universal properties $\mathcal{K}$, written $\Gamma \vdash_{\mathcal{K}} P$, is inductively given by the rules in Table 4.1.*

We now briefly describe each rule, the reader is referred to Section 6.4 for a complete typing example.

- Just like $\mathbf{0}$ is a neutral element of the $\mid$ process constructor (up to $\equiv$, that is) (U-NIL) returns the neutral element of the $\odot$ operator.
- (U-PAR) directly applies the $\odot$ operator (Definition 4.2.6).

- (U-Res) applies a restriction operator (see Definition 4.4.5 below).
- The type of a sum is essentially the types of all its terms connected by disjunction (see Definition 3.6.4), as the process evolves like one of its components. In rule (U-Sum) we also apply the elementary sum rules in the local component, and permits some weakening through the remote component. As we'll see in the Activeness instance (Section 6) this may permit the elementary sum operator to produce stronger statements.
- The (U-Pre) produces a set of statements that are all valid for the process: first, pick some arbitrary channel type and multiplicities for $G$'s subject $p$, then increase $p$'s multiplicities by 1 or (if $G$ is replicated) $\omega$. Then next three factors refer to the objects and are *bound* with the binding operator $\boldsymbol{\nu}\mathrm{bn}(G)$ after composition. These three terms are respectively the continuation, the local properties as constructed by the elementary rules, and remote behaviour. The remote behaviour $\overline{\sigma}[\tilde{x}]$ plays two roles, respectively through the local and environment components of the instantiated channel $\overline{\sigma}[tf]$. First, it states that the $G$'s communication partner will behave according to the protocol specified in the channel type whenever queries are sent to it. Second, its environment multiplicities set upper bounds on how many times the local side is permitted to use the parameters' ports. The **!** operator is described in Definition 4.4.7 below.

The (U-New) and (U-Pre) rules use *binding* of process types. Binding a name amounts to forcing it not to be used in the environment, i.e. its environment multiplicities are forced to zero and dependencies on its universal resources become $\top$ (vacuously true).

Formally:

**Definition 4.4.5 (Binding)** *On dependencies, $(\bar{\boldsymbol{\nu}}x)\varepsilon$ is the logical homomorphism such that:*

$$(\bar{\boldsymbol{\nu}}x)p_k \overset{\text{def}}{=} \begin{cases} \top & \text{if } \mathrm{n}(p) = x \\ p_k & \text{if } \mathrm{n}(p) \neq x \end{cases}$$

*On behavioural statements, $(\boldsymbol{\nu}x)\,\Xi$ is the logical homomorphism such that:*

$$(\boldsymbol{\nu}x)\,(p_k \triangleleft \varepsilon) = \begin{cases} \top & \text{if } \mathrm{n}(p) = x \\ p_k \triangleleft (\bar{\boldsymbol{\nu}}x)\varepsilon & \text{if } \mathrm{n}(p) \neq x \end{cases}$$

*On multiplicities:*

$$(\boldsymbol{\nu}x)\,(p^m) = \begin{cases} \top & \text{if } \mathrm{n}(p) = x \\ (p^m) & \text{if } \mathrm{n}(p) \neq x \end{cases}$$

*Binding a name $x$ in a process type $\Gamma$ is then as follows:*

$$(\boldsymbol{\nu}x)\;(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \overset{\text{def}}{=} \big(\Sigma|_{\mathrm{dom}(\Sigma)\setminus x}; (\boldsymbol{\nu}x)\,\Xi_{\mathrm{L}} \blacktriangleleft (\boldsymbol{\nu}x)\,\Xi_{\mathrm{E}}\big)$$

When typing a replicated process $!\,a(\tilde{y}).P$ or $!\,(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle.P$, $\#(G)$ is $\omega$ and parts of the type related to parameters or to the continuation must be *replicated.*

**Lemma 4.4.6 (Existence of Replication)** *Let $\Gamma$ be a process type. Then there is a natural number $n$ such that either $\Gamma^n$ is not well defined or $\Gamma^n \cong \Gamma^{n+1}$ (where $\Gamma^1 \overset{\text{def}}{=} \Gamma$ and $\Gamma^{n+1} \overset{\text{def}}{=} \Gamma^n \odot \Gamma$).*

We omit the proof but it can be shown in two parts: If $\Gamma$ doesn't use disjunction then $n = 2$ satisfies the requirements (although $n = 1$ may also work for some processes) and $\Gamma^2$ replaces all non-zero multiplicities by $\star$. Secondly, if $\Gamma$'s normal form contains $m$ $\vee$-separated terms then $n = 2m$ satisfies the requirements, as this gives a chance for a dependency chain traversing all $m$ terms to be reduced (and the factor 2 sets multiplicities to $\star$ as before). This is shown by induction on $m$.

This lemma implies that repeatedly composing a process type with itself eventually stabilises, which gives a practical way to compute "$\Gamma^\infty$".

**Definition 4.4.7 (Process Type Replication)** *Replication of a process type $\Gamma$ is defined with the following rule: $\mathbf{!}\,\Gamma \overset{\text{def}}{=} \Gamma^n$ where $n$ is a value satisfying the lemma above.*

In the type system, "$\mathbf{!}_{\text{if something}}\Gamma$" stands for $\mathbf{!}\,\Gamma$ if "something" is true, otherwise it is equal to just $\Gamma$.

For instance when typing $\mathbf{!}\,\overline{a}\langle b\rangle.c$ we obtain the multiplicities $\bar{a}^\omega \odot \mathbf{!}\,(\bar{b}^1 \odot c^1) = \bar{a}^\omega \wedge \bar{b}^\star \wedge c^\star$, assuming $a$ has the usual input-output-alternating type.

## 4.5 Verifying Protocols

The framework and type system given so far assume guards will respect the protocols described in channel types. Let channel $a$ have type $\sigma = ((); \bar{1}_k^\star; \bar{1}_k)$ requiring its parameter to satisfy some property $k$. Then the above system is only sound whenever all $a$-inputs and outputs provide $k$ on their parameter. Consider a process $x.Q$ where $x_k$ depends on some $\varepsilon$. Then, when typing $a(x).x.Q$, the prefix rule computes $(\boldsymbol{\nu} x)\,(x_k \triangleleft \varepsilon) = \top$ (plus some other statements related to $a$ itself), so $\varepsilon$ is lost.

One way to deal with this issue is to include *responsiveness* into $\mathcal{K}$ (Section 4.8, that keeps track of dependencies required by processes to satisfy the protocol, but requires some changes in the type algebra. We will explore this direction in detail in the next section.

Another, simpler but sufficient for universal resources, way is to have a single global "correctness resource" $\mathsf{proc}_{\mathbf{ok}}$ whose elementary rule is given by:

$$\mathsf{prop}_{\mathbf{ok}}(\sigma, G, m, m') = \mathsf{proc}_{\mathbf{ok}} \triangleleft \begin{cases} \sigma[\mathsf{obj}(G)] & \text{if } G \text{ is an input} \\ \overline{\sigma}[\mathsf{obj}(G)] & \text{if } G \text{ is an output} \end{cases} \tag{4.10}$$

and include **ok** into $\mathcal{K}$. Then a typed process $(\Gamma; P)$ satisfies all protocols on its channels if $\Gamma \vdash_{\mathcal{K}} P$ implies $\mathsf{proc}_{\mathbf{ok}} \succeq \Gamma$. The semantic predicate $\mathsf{good}_{\mathbf{ok}}$ is defined to be always true.

## 4.6 Properties

This section summarises the properties enjoyed by the type system.

The following lemma follows from the type system rules being syntax directed:

**Lemma 4.6.1 (Decidability)** *Typability with with respect to a set of universal properties is decidable.*

Structurally congruent processes can be typed the same way (which is one reason processes can safely be identified up to structural congruence):

**Proposition 4.6.2 (Subject Congruence)** *Let $\Gamma \vdash_{\mathcal{K}} P \equiv P'$. Then $\Gamma' \vdash_{\mathcal{K}} P'$ for some $\Gamma' \cong \Gamma$.*

As far as typability is concerned, the transition operator correctly predicts the evolution of a process. If $\mu = \tau$ then $\Gamma \wr \mu = \Gamma$ and this proposition shows that the type of a process remains valid when the process is reduced.

**Proposition 4.6.3 (Subject Reduction)** *Let $(\Gamma; P)$ be a typed process such that $\Gamma \vdash_{\mathcal{K}} P$ with $\mathbf{ok} \in \mathcal{K}$ and $\mathsf{proc}_{\mathbf{ok}} \succeq \Gamma$. Then, for any transition $(\Gamma; P) \xrightarrow{\mu} (\Gamma \wr \mu; P')$, $\exists \Gamma'$ s.t. $\Gamma' \preceq \Gamma \wr \mu$ and $\Gamma' \vdash_{\mathcal{K}} P'$.*

The proof (for the general type system including existential resources and *events* — see the next section) is given in Appendix A.2.

**Conjecture 4.6.4 (Type Soundness)** *If $\Gamma \vdash_{\mathcal{K}} P$ with $\mathbf{ok} \in \mathcal{K}$ and $\mathsf{proc}_{\mathbf{ok}} \succeq \Gamma$, then $\Gamma \models_{\mathcal{U}} P$.*

*Proof* Subject Reduction implies that if $(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'; P')$ then $\Gamma' \succeq \Gamma''$ for some $\Gamma''$ with $\Gamma'' \vdash_{\mathcal{K}} P'$. The semantic predicates must be preserved by weakening, by the definition of elementary rules, so it is enough to show that $(\Gamma''; P')$ is immediately correct, which implies immediate correctness of $(\Gamma'; P')$, and therefore correctness $(\Gamma; P)$.

Due to time constraints I was unable to prove the general case and will instead show immediately correctness of individual instances (Section 8). □

## 4.7 Type System Tuning

Examining the type system, one can notice that two rules are not completely specified, namely (U-PRE) does not specify how to obtain $m$ and $m'$. No matter how these are obtained the type system is sound, and so we left them unspecified to permit some tuning of the type system behaviour. We suggest a few ways of choosing them.

Consider the process $P = a.b|\bar{a}$.

- The simplest way is to always set $m = m' = \star$ in (U-PRE), but since the environment is given lots of freedom, processes can only be given few guarantees. For instance in $P$, none of $a$, $\bar{a}$ or $b$ are active, as any attempt to access any of them could be broken by a third-party process $\big((a^1 \wedge \bar{a}^1 \wedge b^1 \wedge b^0 \blacktriangleleft a^\star \wedge \bar{a}^\star \wedge b^\star \wedge \bar{b}^\star\big) \vdash P)$
- The other extreme is to run the type system twice, first to record the multiplicities obtained in $\Xi_{\mathrm{L}}$, and then using those as values for $m$ and $m'$ in the second run. This basically gives the environment as little permissions as possible, actually so little that in $P$, none of $a$, $\bar{a}$ or $b$ are active because the environment is not permitted to access them $\big((a^1 \wedge \bar{a}^1 \wedge b^1 \wedge \bar{b}^0 \blacktriangleleft a^0 \wedge \bar{a}^0 \wedge b^0 \wedge \bar{b}^0\big) \vdash P)$

- A more interesting middle-ground is to do the above but replacing any $p^\omega \wedge \bar{p}^m$ by $p^\omega \wedge \bar{p}^\star$, and $p^1 \wedge \bar{p}^0$ by $p^1 \wedge \bar{p}^1$. Now in $P$, both $a$ and $b$ are assumed linear, and $b$ is found active: $\left(a^1 \wedge \bar{a}^1 \wedge b^1_{\mathbf{A}} \wedge \bar{b}^0 \blacktriangleleft a^0 \wedge \bar{a}^0 \wedge b^0 \wedge \bar{b}^1\right) \vdash P$.

No matter which of the above variant is chosen it may at times be desirable to override the default behaviour, for instance through annotations in the process.

## 4.8 Responsiveness

As a preamble to the following section on existential properties we now present *responsiveness*, a universal property denoted $\mathbf{R}$ of particular importance when studying existential properties. It permits estimating the dependencies of bound names without keeping track of them individually, and plays a central role in detecting circular forwarding.

In a word, a port $p$ is *responsive* (written $p_{\mathbf{R}}$) in process $P$ if all $p$-prefixes obey $p$'s protocol, in the sense that they provide on their parameters all resources declared in the channel type, with no additional dependencies.

Suppose the (existential) channel property $\mathbf{O}$ stands for "will eventually be subject of an output", and let $\sigma$ be a channel type whose input must provide $\mathbf{O}$ on the parameter:

$$\sigma = \left((); \bar{1}^1 \wedge 1_{\mathbf{O}}; 1^1\right)$$

Then in $P = a(x).\bar{x}$, $a$ is responsive, as passing any name $b$ to $a$ will trigger the output $\bar{b}$, i.e. $P$ provides $b_{\mathbf{O}}$ in response to a request $a(b)$. In $P' = a(x).t.\bar{x}$, where $t$ is linear and $a$ has type $\sigma$, $a$'s responsiveness depends on $t_{\mathbf{O}}$: if $t$ is eventually used in output then $a$ will eventually provide an output at its parameter $x$.

Inversely, let $Q$ be a process doing the transition $Q \xrightarrow{a(b)} Q'$. If $a$ is responsive in $Q$, $b_{\mathbf{O}}$ will be available in $Q'$. If $a$'s responsiveness $a_{\mathbf{R}}$ depends on $\varepsilon$ then $b_{\mathbf{O}}$ also depends on $\varepsilon$ in $Q'$.

In a forwarder $a \gg b$, $a$ can only be responsive if $b$ is, i.e. we have the dependency[3] $a_{\mathbf{R}} \lhd b_{\mathbf{R}}$. When chaining forwarders, dependency gets reduced (Definition 5.1.3), e.g. in $a \gg b \mid b \gg c$, $a$'s responsiveness depends on $c$'s responsiveness. In case of a circular forwarding as in $a \gg b \mid b \gg a$, $a_{\mathbf{R}} \lhd b_{\mathbf{R}} \wedge b_{\mathbf{R}} \lhd a_{\mathbf{R}}$ reduces to $(a_{\mathbf{R}} \wedge b_{\mathbf{R}}) \lhd \bot$, i.e. neither $a$ nor $b$ is responsive in that process.

Semantics of responsiveness is provided by the following definition:

**Definition 4.8.1 (Responsiveness Semantic Predicate)**
*The semantic predicate* $\mathsf{good}_{\mathbf{R}}$ *for responsiveness is such that* $\mathsf{good}_{\mathbf{R}}(p \lhd \varepsilon, (\Gamma; P))$ *if for all transitions* $(\Gamma; P) \xrightarrow{(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle}$ *s.t.* $\mathrm{n}(p) \in \tilde{x} \setminus \tilde{z}$: $\overline{\sigma}[\tilde{x}]|_{p_{\mathbf{R}}} = p_{\mathbf{R}} \lhd \varepsilon_0$ *with* $(a_{\mathbf{R}} \wedge \varepsilon_0) \preceq \varepsilon$ *(*$\sigma$ *being* $a$*'s type according to* $\Gamma$*).*

Why is it enough to only check responsiveness of output objects?

Responsiveness is usually tested in two phases, one to "ask a question" and one for the process to "reply to it" (for instance testing $a$'s responsiveness in the process $a(x).\bar{x}$ is done with the "question" $\xrightarrow{a(x)}$ followed by the "answer" $\xrightarrow{\bar{x}}$). For such tests responsiveness is always "immediately correct" as it takes more than one transition to test it. However when an output process *delegates*

[3] $a_{\mathbf{R}}$ also depends on $b$'s input *activeness*, as we'll see in Section 6 later on.

responsiveness of an object port, a single transition can disprove a responsiveness statement. For instance if a process exhibits the $\xrightarrow{\overline{a}\langle b\rangle}$ transition, assuming one parameter i/o-alternating channel types, we can immediately infer that $b_{\mathbf{R}}$ must depend at least on $a_{\mathbf{R}}$. This is what the above definition checks.

As we will see in the next section, the question-and-reply semantics of responsiveness is actually provided by the transition operator (Definition 5.1.6 on page 50), that (at "question time") converts a responsiveness statements into statements specified by the channel type, which, in turn, are verified by the liveness semantics (Definition 5.2.6 on page 54).

**Definition 4.8.2 (Responsiveness Elementary Guard Rule)** *The dependencies of a responsiveness resource $p_{\mathbf{R}}$ are obtained with the elementary rule*

$$\mathsf{prop}_{\mathbf{R}}(\sigma, G, m, m') = \mathsf{sub}(G)_{\mathbf{R}} \lhd \begin{cases} \sigma[\mathsf{obj}(G)] & \textit{if } G \textit{ is an input} \\ \overline{\sigma}[\mathsf{obj}(G)] & \textit{if } G \textit{ is an output} \end{cases} \tag{4.11}$$

i.e. $p$ is responsive if all resources declared in the channel type are provided.

# Chapter 5

# Existential Properties

This section extends the previous one with existential properties. Remember (Definition 4.1.1, page 31) that an existential property available in a process is also available when that process is composed with another process. As we'll see later, existential properties have liveness semantics.

Two important extensions must be done to the theory presented in the previous section. First, to implement that intuitive definition of "existentialism", the spatial operators and relations ($\odot$, $\hookrightarrow$ and binding, as opposed to logical relations $\cong$ and $\succeq$) use the dual logical connectives when acting on existential resources compared to universal resources. Secondly, the universal type system was treating a process $a.P$ precisely the same way as $a \,|\, P$ (indeed, if something bad can happen in $P$, it is just one $\xrightarrow{a}$-transition away from $a.P$, so that process isn't safe either, unless $a$ is not observable but this is out of the type system's scope), but such a shortcut isn't acceptable for a property with liveness semantics: if something good eventually happens in $P$, we need to be sure $a$ can be consumed before we can state the good thing happens in $a.P$ as well. This manifests itself as an operator for dependencies of a transition (Definition 5.2.5).

Before proceeding to the type algebra we introduce a few restrictions on what channel types are acceptable:

**Definition 5.0.3 (Restrictions on Channel Types)** *Let $\sigma = (\tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}})$.*

*The channel type $\sigma$ is said to have* shared liveness *(between its input and its output) if there is an existential resource $p_k$ such that both $p_k^m \lhd \varepsilon \succeq \xi_{\mathrm{I}}$ and $p_k^{m'} \lhd \varepsilon' \succeq \xi_{\mathrm{O}}$ for some $m$, $m'$, $\varepsilon$ and $\varepsilon'$.*

*The type is said to have* blocked liveness *if there is a port $p$ such that either $p_k^m \lhd \varepsilon \succeq \xi_{\mathrm{I}}$ and $\bar{p}^0 \succeq \xi_{\mathrm{O}}$, or $p_k \lhd \varepsilon \succeq \xi_{\mathrm{O}}$ and $\bar{p}^0 \succeq \xi_{\mathrm{I}}$, with $k \in \mathcal{E}$.*

*We also say it is a case of blocked liveness if there is a term $p_k^0$ in either $\xi_{\mathrm{I}}$ or $\xi_{\mathrm{O}}$.*

*A channel type has* unstable multiplicities *if (at least) one of $\xi_{\mathrm{I}}$ and $\xi_{\mathrm{O}}$ include $\{p^1 \wedge \bar{p}^m\}$, for some $p$ and non-zero $m$.*

Channel types with shared liveness need special care in a type system. Consider for example the channel type

$$\sigma = (()(); \bar{1}^{\star}_{\mathbf{A}} \wedge 1^{\star} \wedge 2_{\mathbf{A}} \lhd \bar{1}_{\mathbf{A}}; \bar{1}^{\star}_{\mathbf{A}} \wedge 1^{\star} \wedge \bar{2}_{\mathbf{A}} \lhd \bar{1}_{\mathbf{A}}) .$$

It has shared liveness on $\bar{1}_{\mathbf{A}}$, and a valid input with $a : \sigma$ is $a(xy).(!\,\bar{x} \mid x.y)$. The process

$$a(xy).x.y \tag{5.1}$$

on its own does not respect the protocol because it does not provide activeness on $\bar{x}$. Similarly, a valid output for the same type is $\overline{a}\langle bc\rangle.(!\,\bar{b} \mid b.\bar{c})$. Note that *both* the input and the output on $a$ are required by the protocol to provide output activeness on the first parameter, which is exactly "shared activeness".

The reason it needs special care in a type system is that a naive treatment would result in (5.1) being accepted: indeed, the protocol requires the output to provide an $x$-output without conditions, and the input in (5.1) can be considered to have delegated its work on $x$ to the $a$-output. Yet of course a similar reasoning would allow the output to delegate its work to the $a$-input, resulting in neither of them doing it. We will see cases where such delegation is acceptable.

Types with blocked liveness simultaneously require one port of the channel to provide some existential resource on a parameter, and forbid the other port to connect to that parameter. The reason for ruling out such types is that analysing processes such as $\overline{a}\langle a\rangle$ with $a : \sigma = (\sigma; \top; \bar{1}^{\star}_{\mathbf{A}} \wedge 1^{\star})$ becomes more difficult — On the one hand the request itself seems to fulfil the protocol, as it is an output on $a$, and on the other hand, as soon as the request is sent the output is no longer available but, simultaneously, the $a$-input is not be permitted to attempt accessing its parameter. Ruling out blocked liveness avoids such paradoxical cases.

Finally, a valid receiver (or sender) on a channel type with unstable multiplicities may become invalid through a $\tau$-transition, by consuming its own parameters. For instance, having $a : \sigma = ((); \ldots; 1 \wedge \bar{1})$, $P = (\boldsymbol{\nu} b)\,(\overline{a}\langle b\rangle \mid b \mid \bar{b})$ is a correct output. But of course $P \ \rightarrow (\boldsymbol{\nu} b)\,(\overline{a}\langle b\rangle)$, which isn't.

Because of that, and because we believe there is little (if any) use to such channel types, we apply, in the rest of this thesis, the following:

**Convention 5.0.4** *No channel types involved in a semantic judgement (Definition 5.2.6) or in a typing judgement (Sections 4.4 and 5.3) may have shared or blocked liveness, or unstable multiplicities (this also applies to parameter types at all depths).*

## 5.1 Existential Type Algebra

From this point on, behavioural statements can use both existential and universal properties, even within a single dependency statement (i.e. a universal resource may depend on an existential one or the other way round). To spare the reader from moving back and forth between this section and the previous one we present the full properties in this section, with both rules given in the previous sections and ones specific to existential properties.

The difference between existential and universal properties is made explicit in the following definition that extends Definition 3.9.3 and rule (4.4) on pages 26 and 33.

**Definition 5.1.1 (Behavioural Statement Composition)** composition *on behavioural statements is given by the logical homomorphism $\odot$ such that:*

1. $(p^{m}) \odot (p^{m'}) \overset{\text{def}}{=} p^{m+m'}$

2. $(p_k \lhd \varepsilon) \odot (p_k \lhd \varepsilon') \overset{\text{def}}{=} (p_k \lhd \varepsilon) \vee (p_k \lhd \varepsilon')$ *if* $k \in \mathcal{U}$.

3. $(p_k \lhd \varepsilon) \odot (p_k \lhd \varepsilon') \overset{\text{def}}{=} (p_k \lhd \varepsilon) \wedge (p_k \lhd \varepsilon')$ *if* $k \in \mathcal{E}$.

4. $\Xi \odot \bot \overset{\text{def}}{=} \bot$

5. *When no other rule applies,* $\Delta \odot \Delta' \overset{\text{def}}{=} \top$.

Point 5 above and Convention 4.2.2 interact in a subtle way to give the following property:

**Lemma 5.1.2 (Composition of disjoint statements)** *For two statements* $\Xi$ *and* $\Xi'$*, having no resources in common when written according to Convention 4.2.2 (specifically, its point 1),* $\Xi \odot \Xi' = \Xi \wedge \Xi'$

The proof is given in Appendix A.1.7 on page 132.

**Definition 5.1.3 (Dependency Reduction)** *The* reduction relation $\hookrightarrow$ *on behavioural statements is a partial order relation satisfying*

1. $(p_k \lhd \varepsilon) \wedge (\gamma \lhd \varepsilon') \hookrightarrow (p_k \lhd \varepsilon) \wedge (\gamma \lhd \varepsilon'\{^{\varepsilon\{^{\bot}/_{\gamma}\} \wedge p_k}/_{p_k}\})$ *for* $k \in \mathcal{U}$.

2. $(p_k \lhd \varepsilon) \wedge (\gamma \lhd \varepsilon') \hookrightarrow (p_k \lhd \varepsilon) \wedge (\gamma \lhd \varepsilon'\{^{\varepsilon\{^{\bot}/_{\gamma}\} \vee p_k}/_{p_k}\})$ *for* $k \in \mathcal{E}$

   *On process types:*

3. $\Xi \hookrightarrow \Xi'$ *implies* $(\Xi \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow (\Xi' \blacktriangleleft \Xi_{\mathrm{E}})$ *and* $(\Xi_{\mathrm{L}} \blacktriangleleft \Xi) \hookrightarrow (\Xi_{\mathrm{L}} \blacktriangleleft \Xi')$.

4. $(\gamma_k \lhd \varepsilon_1 \blacktriangleleft \gamma_k \lhd \varepsilon_2) \hookrightarrow (\gamma_k \lhd (\varepsilon_1 \wedge \varepsilon_2) \blacktriangleleft \gamma_k \lhd \varepsilon_2)$ *for* $k \in \mathcal{U}$.

5. *If* $(\alpha \lhd \varepsilon) \preceq \Xi_{\mathrm{E}}$ *with* $\beta \preceq \varepsilon$ *then* $(\gamma \lhd \varepsilon' \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow \big(\gamma \lhd (\varepsilon'\{^{\alpha \wedge \beta}/_{\alpha}\}) \blacktriangleleft \Xi_{\mathrm{E}}\big)$ *for* $\beta \neq \gamma$.

6. *If* $(\Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow (\Xi'_{\mathrm{L}} \blacktriangleleft \Xi'_{\mathrm{E}})$ *then* $(C[\Xi_{\mathrm{L}}] \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow (C[\Xi'_{\mathrm{L}}] \blacktriangleleft \Xi'_{\mathrm{E}})$ *and* $(\Xi_{\mathrm{L}} \blacktriangleleft C[\Xi_{\mathrm{E}}]) \hookrightarrow (\Xi'_{\mathrm{L}} \blacktriangleleft C[\Xi'_{\mathrm{E}}])$ *for any* local context[1] $C[\cdot]$.

*A behavioural statement* $\Xi$ *is* closed *if* $\Xi \hookrightarrow \Xi'$ *implies* $\Xi \cong \Xi'$. *A* closure *of a behavioural statement* $\Xi$*, written* close $(\Xi)$*, is* $\Xi'$ *such that* $\Xi \hookrightarrow \Xi'$ *and* $\Xi'$ *is closed.*

Note again the difference in treatment of existential and universal resources, that is very similar to the one occurring in behavioural statement composition. Indeed it is a simple exercise to verify that dependency reduction commutes with composition or, more accurately:

**Lemma 5.1.4** *For any two behavioural statements* $\Xi$ *and* $\Xi'$*:*

$$\operatorname{close}(\operatorname{close}(\Xi) \odot \Xi') \cong \operatorname{close}(\Xi \odot \Xi')\,.$$

[1]I.e. $C \;::=\; [\cdot] \;\mid\; C \wedge \Delta \;\mid\; C \vee \Delta$

Note also how self-dependencies $\gamma \lhd \gamma$ are not permitted and replaced by $\gamma \lhd \bot$ in both cases. Existential self-dependencies are for example found in deadlocks such as $a.\bar{b}|b.\bar{a}$ where $\bar{a}_{\mathbf{A}}$ and $\bar{b}_{\mathbf{A}}$ depend on each other, and responsiveness self-dependencies are for example found in forwarder loops (sometimes called *livelocks* in the literature) such as $!\,a(x).\overline{b}\langle x\rangle \mid !\,b(x).\overline{a}\langle x\rangle$ where $a_{\mathbf{R}}$ and $b_{\mathbf{R}}$ depend on each other. We'll see in Section 5.5 some cases of self-dependencies that aren't similarly harmful, and how to deal with them nicely.

Definition 4.2.5 (specifically, its second point) is generalised to existential resources as expected:

**Definition 5.1.5 (Removing Non-Observable Dependencies)** *Let $\Gamma$ be a process type.* Removing *non-observable dependencies in it is done by the* clean *operator, applying the following operations on its local behavioural statement $\Xi_{\mathrm{L}}$ as many times as possible:*

- *Replace any statement $p_k \lhd \varepsilon$ where $p$ is not observable (Definition 3.7.2) in $\Gamma$ by $\top$*
- *In any statement $\gamma \lhd \varepsilon$, for any $p$ not observable in $\Gamma$'s complement $\overline{\Gamma}$, replace any $p_k$ $(k \in \mathcal{U})$ in $\varepsilon$ by $\top$,* and *any $p_k$ $(k \in \mathcal{E})$ in $\varepsilon$ by $\bot$.*

The transition operator (Definition 4.2.10) is modified to use remote responsiveness when computing dependencies of remote resources.

**Definition 5.1.6 (Transition Operator)** $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *being a process type with $\Sigma(a) = \sigma$, the effect of a transition $\mu$ on $\Gamma$ is $\Gamma \wr \mu$, defined as follows.*

- $\Gamma \wr \tau \overset{\text{def}}{=} \Gamma$,
- $\Gamma \wr a(\tilde{x}) \overset{\text{def}}{=} \Gamma \wr a \odot \sigma[\tilde{x}] \lhd (a_{\mathbf{R}} \blacktriangleleft \bar{a}_{\mathbf{R}}) \odot \mathsf{prop}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle, \sigma, m, m')$,
- $\Gamma \wr (\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\,\overline{a}\langle\tilde{x}\rangle \overset{\text{def}}{=} \Gamma \wr \bar{a} \otimes \bar{\sigma}[\tilde{x}] \lhd (\bar{a}_{\mathbf{R}} \blacktriangleleft a_{\mathbf{R}}) \odot \mathsf{prop}_{\mathcal{K}}(a(\tilde{x}), \sigma, m, m')$.

In the above definition, $\Gamma \lhd (\bar{a}_{\mathbf{R}} \blacktriangleleft a_{\mathbf{R}})$ makes $\Gamma$'s local component depend on $\bar{a}_{\mathbf{R}}$ and its environment component depend on $a_{\mathbf{R}}$.

Point 4 from Definition 5.1.3 now becomes important for removing remote behaviour from the type. For instance $\overline{a}\langle x\rangle.x.\bar{s}$, where $a$ is alternating, may have reduced $\bar{s}_{\mathbf{A}} \lhd \bar{x}_{\mathbf{A}}$ and $\bar{x}_{\mathbf{A}} \lhd a_{\mathbf{R}}$ into $\bar{s}_{\mathbf{A}} \lhd a_{\mathbf{R}}$. Simulating the $\overline{a}\langle x\rangle$ transition effectively cancels the $\bar{x}_{\mathbf{A}} \lhd a_{\mathbf{R}}$ term and the reduction it caused, as the environment component of $\overline{\sigma}[x] \lhd (\bar{a}_{\mathbf{R}} \blacktriangleleft a_{\mathbf{R}})$ contains $x_{\mathbf{A}} \lhd a_{\mathbf{R}}$ which, through rule 4, replaces $a_{\mathbf{R}}$-dependencies by $\bar{x}_{\mathbf{A}}$ dependencies. Similarly, the $\bar{x}_{\mathbf{A}} \lhd a_{\mathbf{R}}$ statement becomes $\bar{x}_{\mathbf{A}} \lhd \bar{x}_{\mathbf{A}}$, i.e. $\bar{x}_{\mathbf{A}} \lhd \bot$.

## 5.2 Existential Semantics

While universal properties enjoy safety semantics, existential properties enjoy *liveness* semantics, but here again not all liveness semantics can be expressed with existential properties, a counter-example being termination (specifically, having no infinite sequences of $\tau$-reductions: $!\,a$ and $!\,\bar{a}$ terminate but their composition $!\,a \mid !\,\bar{a}$ doesn't). A statement $\gamma \lhd \top \models P$, where $\gamma$ is an existential resource, guarantees that $\gamma$ *will eventually be provided by $P$.*

Just like safety, semantics of an existential property is parametrised by a predicate $\mathsf{good}_k$, which must match the following definition:

**Definition 5.2.1 (Existential Predicate, Success State)**
*A semantic predicate* $\mathsf{good}_k$ *is* existential *if, whenever* $\Gamma \odot \Gamma'$ *is well-defined,* $\mathsf{good}_k(p \lhd \top, (\Gamma; P))$ *implies* $\mathsf{good}_k(p \lhd \top, (\Gamma \odot \Gamma'; P \mid P'))$.

*A typed process* $(p_k \lhd \varepsilon; P)$ *is said in a* success state *if* $\mathsf{good}_k(p \lhd \varepsilon, (\Gamma; P))$ *is true.*

A naive definition of liveness would be just replacing "for all transition sequences" by "there is a transition sequence", in Definition 4.3.4. However, in order for such a property to be useful we need a degree of reliability (which is implied by the future tense and the word "eventually"), as the scheduler might not necessarily follow that exact sequence.

The word "eventually" also hides a *fairness* assumption on the scheduler, which says, for a scheduler, that if a particular transition is constantly available, it will eventually occur. Instead of a particular transition we shall use a *strategy function*, but let's not get ahead of ourselves. We now consider a series of processes $P_i$ with increasing requirements for liveness, and, for each, we discuss whether a resource $\gamma$ should be considered to be "eventually provided". This will help indicating more accurately what is required of the scheduler.

To keep this discussion general, assume that some process $S$ immediately provides some existential resource $\gamma$ (i.e. $(\gamma; S)$ is in a success state). For instance $S$ could be a success signal $!\,\bar{s}$ and $\gamma$ is immediately available in a process if it has an $\bar{s}$ barb.

The simplest example is $P_1 = S$. If $\gamma$ is immediately available, then it is also eventually available.

Now an unrelated but unending computation should not affect liveness: In $P_2 = S|\Omega$, $\gamma$ is still available, $\Omega$ being some process with $\Omega \;\rightarrow \Omega$, for instance

$$\Omega \overset{\text{def}}{=} (\boldsymbol{\nu} t)\,(\bar{t} \mid !\,t.\bar{t}) \tag{5.2}$$

A finite number of transitions preserves liveness (but not *immediate* availability). Resource $\gamma$ is eventually provided in $P_3 = \tau.\tau \ldots \tau.S \mid \Omega$ with a finite number of $\tau$, and where

$$\tau.P \overset{\text{def}}{=} (\boldsymbol{\nu} t)\,(\bar{t}|t.P) \tag{5.3}$$

for some fresh $t$.

The following example is more interesting in that the number of transitions is no longer bounded:

$$P_4 = !\,a(x).\bar{x} \mid !\,a(x).\bar{a}(\boldsymbol{\nu} y).y.\bar{x} \mid \bar{a}(\boldsymbol{\nu} t).t.S$$

In this process, every request sent to $a$ may be received by the first or by the second input. The first input immediately responds to requests while the second one resends the request. So, the request $\bar{a}(\boldsymbol{\nu} t)$ will, under a fair scheduler, loop in the second input for a while, and then eventually be passed to the first input, after which the requests to the second input cascade back until the $\bar{t}$ output is fired and $S$ freed. It seems therefore reasonable to consider $\gamma$ to be eventually provided by this process, as it matches the accepted definition of fairness (see for instance [PT00], Section 2.9, as well as [CC04]), in that the first input is continuously available, and at each step there is a request to $a$ available, so that the first input should eventually catch one request, after which we are back to $P_3$.

A last example, which, in our opinion, would be requiring too much from a scheduler (as even a stochastic scheduler would not satisfy it), is the following. This example shows that a naive liveness definition such as "$\forall Q$ s.t. $P \Rightarrow Q$, $\exists R$ s.t. $Q \Rightarrow R$ and $(\gamma; R)$ is in a success state" would be too weak.

$$P_6 = \mathbf{!}\, a(x).\bar{x} \mid \mathbf{!}\, a(x).\overline{a}(\boldsymbol{\nu} y).y.\overline{a}\langle x\rangle \mid \overline{a}(\boldsymbol{\nu} t).t.S \tag{5.4}$$

In this example ($P_5$ comes later — larger numbers correspond to increasing requirements on the scheduler), the second output, when receiving the reply to its own request $\overline{a}\langle y\rangle$, re-sends the request it received rather than replying to it, so that the global behaviour is analogous to a random walk. Although a stochastic scheduler (randomly and independently choosing one $a$-input for each $a$-output) will eventually reach $S$, adding more copies of the second input to the program will have the probability of $\gamma$ becoming available fall to zero.

The fundamental difference between $P_4$ and $P_6$ is that in the former, at any point, there is a possibility of *progress* towards immediate availability of $\gamma$. In other words, at any time, there exists a *strong* transition that brings the process "closer" to $S$. In $P_4$ this progress is very simple, in that having the first input handle a request passes from a process where the number of required $\tau$-transitions is not bounded, to one where it is bounded. A process $P_4'$ where the progress is slightly more elaborate would be obtained by replacing $\overline{a}(\boldsymbol{\nu} t)$ in that process by $\overline{a}\langle t_1\rangle.t_1.\overline{a}\langle t_2\rangle.t_2 \ldots t_{n-1}.\overline{a}(\boldsymbol{\nu} t_n)$. In that case, the "distance" towards an output at $s$ is $n$, and is reduced by one every time the first $a$-input is used. When that distance reaches zero, we are back to case $P_3$, with a finite number of transitions. The usual fairness assumption now works, because if at any point in time the scheduler has the possibility to make an (irreversible) progress towards a success state, and if the number of times such progress is required is bound (it is 1 for $P_4$ and $n$ for $P_4'$), then $S$ will eventually be reached. In $P_6$, no such irreversible progress occurs, because any diminution of the call stack can be cancelled by calling the second $a$-input a sufficient number of times.

In order to obtain a precise definition for liveness we introduce a "game" between two players (The "1"-prefix will become clear later in this section).

**Definition 5.2.2 (1-Liveness)** *An existential property $\gamma$ is 1-*eventually *provided by $P$ if Player 2 has a winning strategy in the following game (where "current process" is initially $P$):*

*Player 1 plays first, and, at each turn, may replace the current process $P'$ with any process $Q$ such that $P' \Rightarrow Q$.*

*Player 2, at each turn, may either do nothing or replace the current process $P'$ with any process $Q$ such that $P' \xrightarrow{\tau} Q$.*

*Player 2 has won if the current process $P'$ is ever in a success state.*

In that definition, Player 2 models the "opportunities" the scheduler has to make progress, while player 1 models the times when the scheduler doesn't "take advantage" of those opportunities.

It is now clear that, in $P_4$, player 2 simply connects any existing $a$-output to the first input, and wins, while, in $P_6$, player 1 simply activates the second input at least once at every turn, preventing $S$ to ever become available.

Although we are getting close, this definition is still not good enough.

For instance it does not consider $\gamma$ eventually available in the process

$$P_5 = \mathbf{!}\, a.(\boldsymbol{\nu} t)\,(t \mid \bar{t}.\bar{a}) \mid \mathbf{!}\, a.S \mid \bar{a}$$

An infinite transition sequence always picking the $a$-input on the left, and only letting the strategy do the communication on $t$, satisfies the requirements in Definition 5.2.2, without ever bringing $S$ to top-level.

For the same reasons explained above we believe this process should also be accepted (also compare with the very similar process $\mathbf{!}\,a.\bar{a} \mid \mathbf{!}\,a.S \mid \bar{a}$ where $\gamma$ is recognised as eventually available by both semantics).

We therefore refine the game by permitting player 2 to play more than one transition.

**Definition 5.2.3 ($n$-Liveness, Liveness)** *A resource $\gamma$ will* $n$-eventually *be provided by a process if it satisfies the definition above but where player 2 is allowed up to $n$ transitions. $\gamma$ is* eventually *available whenever it is $n$-eventually available for some $n$.*

With this definition $\gamma$ is eventually provided by $P_i$ iff $i < 6$.

Now that we have a good definition of "eventually" we can define correctness of a full typed process.

A natural semantic definition of a dependency statement $\delta_1 \lhd \delta_2$ for a typed process $(\Gamma_1; P)$ would be "for all correctly typed processes $(\Gamma_2; P_2)$ such that $\delta_2$ is included in $\Gamma_2$ and $\Gamma_1 \odot \Gamma_2$ is well defined, $(\Gamma_1 \odot \Gamma_2; P_1 \mid P_2)$ satisfies $\delta_1$."

That definition happens to be very difficult to work with, mainly because of the universal quantification on $P_2$. Just as it is common to use labelled bisimulations instead of barbed equivalences we use a definition based on labelled transitions.

Assuming an elementary statement $\bigvee_i \left(\gamma_i \lhd \bigwedge_j \alpha_{ij}\right)$ is satisfied by a process, there must be a "path" in the transition network that uses no more external resources than declared in the statement, and that "leads to" a set of processes where one of the $\gamma_i$ is immediately available. We call such a path a *strategy* (in Definition 5.2.2 it represents a strategy for player 2).

**Definition 5.2.4 (Strategy Function)** *A* strategy function *$f$ is a function mapping typed processes to pairs of transition labels and typed processes s.t. if $f(\Gamma; P) = (\mu; \Gamma'; P')$ then $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$.*

*We also define a relation $\xrightarrow{f}$ where $f(\Gamma; P) = (\mu; \Gamma'; P')$ implies $(\Gamma; P) \xrightarrow{f} (\Gamma'; P')$, and $(\Gamma; P) \xrightarrow{f} (\Gamma; P)$ otherwise (if $(\Gamma; P)$ is not in $f$'s domain).*

In other words, whenever a typed process is missing from a strategy's domain, it means that the strategy is to leave the process unchanged rather than performing a transition. When constructing a strategy function we exclude typed processes containing statements that are immediately correct from the function's domain.

For a strategy $f$ to prove a statement $\gamma \lhd \varepsilon$, it should not use more resources than declared in $\varepsilon$.

The following operator makes precise what resources are needed to perform a transition (and are property-dependant).

**Definition 5.2.5 (Dependencies of a Transition)** *The* dependency operator *of an existential property $k$ is a function $\mathsf{dep}_k$ mapping transition labels $\mu$ to dependencies $\mathsf{dep}_k(\mu)$, that commutes with substitution ($\mathsf{dep}_k(\mu\{^{\tilde{x}}/_{\tilde{y}}\}) = \mathsf{dep}_k(\mu)\{^{\tilde{x}}/_{\tilde{y}}\}$) and maps $\tau$ to $\top$.*

A typical definition will be $\mathsf{dep}_k(\tau) = \top$, and $\mathsf{dep}_k(\mu)$ with $\mathsf{sub}(\mu) = p$ to be availability of a $\bar{p}$-prefix. When using many existential properties $k_1$, $k_2$, $\ldots$, use $\mathsf{dep}_{k_1}(\mu) \wedge \mathsf{dep}_{k_2}(\mu) \wedge \ldots$. Indeed, that operator will only ever be used as $\bigwedge_{k\in\mathcal{K}} \mathsf{dep}_k(\mu)$ so we shall use the abbreviation $\mathsf{dep}_{\mathcal{K}}(\mu)$.

Should an application require $\mathsf{dep}_k(\tau)$ to be a dependency not $\cong$-equivalent to $\top$ for at least one $k \in \mathcal{E}$, the dependency reduction relation (Definition 5.1.3) must be altered as follows:

$$(p_k \triangleleft \varepsilon) \wedge (\gamma \triangleleft \varepsilon') \hookrightarrow (p_k \triangleleft \varepsilon) \wedge (\gamma \triangleleft \varepsilon' \{^{(\mathsf{dep}_{\mathcal{K}}(\tau) \wedge \varepsilon)\{^{\perp}/_{\gamma}\} \vee p_k}/_{p_k}\})$$

The actual liveness semantic definition generalises *fairness* (Definition 5.2.2) to labelled transitions and arbitrarily complex behavioural statements.

**Definition 5.2.6 (Existential Semantics)** *Let $\Gamma$ be a type and $P$ a process. We say that $P$* satisfies *$\Gamma$ (or $\Gamma$ is* correct *for $P$), written $\Gamma \models P$, if there is a strategy $f$ satisfying the following.*

*For any infinite sequence of the form $(\Gamma; P) = (\Gamma_0; P_0) \xrightarrow{\tilde{\mu}_0}\searrow (\Gamma'_0; P'_0) \xrightarrow{f} (\Gamma_1; P_1) \cdots \xrightarrow{\tilde{\mu}_i}\searrow (\Gamma'_i; P'_i) \xrightarrow{f} (\Gamma_{i+1}; P_{i+1}) \cdots$, such that, $\forall i, j$: $i < j$ implies $\Gamma'_i \preceq \Gamma'_j$ and $\forall i > 0$: $\tilde{\mu}_i$'s input objects are all fresh and distinct. Let (for all $i$) $\mu_i$ be the label of the $(\Gamma'_i; P'_i) \xrightarrow{f} (\Gamma_{i+1}; P_{i+1})$ transition (or "$\tau$" if it is the identity).*

*Then there is a resource $p_k$ and a number $n$ such that:*

1. *for all $i$, $(p_k \triangleleft \mathsf{dep}_{\mathcal{K}}(\mu_i)) \preceq \Gamma'_i$*

2. *For some $\varepsilon$ with $(p_k \triangleleft \varepsilon) \preceq \Gamma_n$, $\mathsf{good}_k(p \triangleleft \varepsilon, (\Gamma_n; P_n))$.*

Although the $\{P_i\}$ sequence is infinite, it may correspond to a finite number of (strong) transitions if, after some point, all $\tilde{\mu}_i$ are empty and the strategy does no transition.

Note that this definition coincides with Definition 5.2.2 if $\Gamma$'s local component contains a single statement $\gamma \triangleleft \top$ and $\mathsf{dep}_{\mathcal{K}}(\mu) = \top$ iff $\mu = \tau$. Technically this should be called "1-correctness" as the strategy is allowed a single transition at a time, but it so happens that our soundness theorem holds for this definition of correctness, making it a stronger result than soundness for "$n$-correctness".

Remember (rule (4.1) on page 32) that disjunctions on the dependency side can be passed on the other side of the $\triangleleft$ connective, where they become conjunctions, which can then be dropped through projection. For example: $\gamma \triangleleft (\alpha_1 \vee \alpha_2) \cong (\gamma \triangleleft \alpha_1) \wedge (\gamma \triangleleft \alpha_2) \searrow (\gamma \triangleleft \alpha_i)$ for any $i \in \{1, 2\}$. Because of this, the $\searrow$ relation precisely characterises the environment's freedom in resource negotiation. Assume a process has a local component $\gamma_1 \triangleleft (\alpha_{11} \vee \alpha_{12}) \wedge \gamma_2 \triangleleft (\alpha_{21} \vee \alpha_{22})$. It has four projections, one of which is $\gamma_2 \triangleleft \alpha_{21}$, which corresponds to the environment requesting $\gamma_2$, and providing $\alpha_{21}$ in exchange.

While projections deal with disjunctions on the right of the $\triangleleft$ connective, disjunctions on its left need to be handled specially. Note how the statement $(\Xi_1 \vee \Xi_2) \models P$ is strictly weaker (in a logical sense) than $(\Xi_1 \models P) \vee (\Xi_2 \models P)$, for reasons analogous to the modal logic statement $\Box(\Xi_1 \vee \Xi_2)$ being weaker than $(\Box\Xi_1) \vee (\Box\Xi_2)$: it could be that the selection has not yet been made in $P$, and will only occur after a few transitions. Because of that we can't define the semantics of a disjunction in terms of the semantics of the individual terms. On the other hand $(\Xi_1 \wedge \Xi_2) \models P$ is equivalent to $(\Xi_1 \models P) \wedge (\Xi_2 \models P)$, just like $\Box(\Xi_1 \wedge \Xi_2) \iff (\Box\Xi_1) \wedge (\Box\varepsilon_2)$ in most modal logics.

This is addressed in Definition 5.2.6 by first picking a full transition sequence and *then only* by requiring the outcome of the selection to be decided, which can be seen in the definition in the expression "there is $p_k$ such that...". Note how the transition sequence interleaves single invocations of the strategy between arbitrarily long transition sequences, resulting in what we believe to be a good characterisation of fairness. The "eventually" aspect of activeness is covered by the "there is $n$ s.t.".

Passing non-fresh names to inputs may connect two otherwise unrelated parts of the process. This is also modelled by dependency reduction between the $\overline{\sigma}[\tilde{x}]$ term and the rest of the type performed by the $\odot$ operator, as described in Definition 5.1.6.

The weakening constraint "$i < j$ implies $\Gamma'_i \preceq \Gamma'_j$" is a compact way of requiring a particular transition sequence not to "change its mind" on what is being requested. The first sequence $\tilde{\mu}_0$ is unrestricted and may pick any part of the process type after any kind of interference, but in subsequent transitions, new statements introduced by the $\wr$ operator through the $\sigma[\tilde{x}]$ factor must be discarded with the $\searrow$ relation in order to satisfy the weakening constraint.

The following formalises the intuition behind weakening:

**Lemma 5.2.7 (Bisimulations and Type Equivalence)** *Let a typed process $(\Gamma; P)$ be such that $\Gamma \models P$. Then, for any $\Gamma' \succeq \Gamma$ and any $P' \sim P$, if $\Gamma' \models_{\#} P'$ then $\Gamma' \models P'$.*

See Appendix A.1.8 for the proof.

We need the simple-correctness check because uniformity is not always preserved by bisimilarity. On the other hand we have the following corollary which justifies our identifying types up to $\cong$ and processes up to $\equiv$. See also Proposition 5.6.2.

**Corollary 5.2.8** *Let $\Gamma \cong \Gamma'$ and $P \equiv P'$. Then $\Gamma \models_{\#} P$ if and only if $\Gamma' \models_{\#} P'$, and $\Gamma \models P$ if and only if $\Gamma' \models P'$.*

## 5.3 Existential Type System

In this section we will extend the type system given in Section 4.4 to work with existential properties, i.e. for any $\mathcal{K}$ with $\{\mathbf{R}\} \subseteq \mathcal{K} \subseteq \mathcal{U} \cup \mathcal{E}$.

Just like the universal type system, this one is parametrised with guard and sum elementary rules for each $k \in \mathcal{K}$. Assume for now the elementary rules for existential properties are *local* (Definition 4.4.2), and refer to Section 6.5 for the additional requirements for soundness and an example.

Given a process $P$, a mapping $\Sigma$ of channel types for all free names, and optionally multiplicities for some names, the type system constructs a process type $\Gamma$ for $P$. Processes deemed unsafe (that may violate multiplicity constraints or mismatch channel types) are rejected as untypable. Incompleteness means that typing may fail for process that are actually safe, and even when typing succeeds, the behavioural statement constructed by the type system may be weaker than what is actually satisfied by the type system.

$$\frac{-}{(\varnothing; \top \blacktriangleleft \top) \vdash_{\mathcal{K}} \mathbf{0}}\ \text{(E-Nil)}$$

$$\frac{\forall i : \Gamma_i \vdash_{\mathcal{K}} P_i}{\Gamma_1 \odot \Gamma_2 \vdash_{\mathcal{K}} P_1 \,|\, P_2}\ \text{(E-Par)} \qquad \frac{\Gamma \vdash_{\mathcal{K}} P \qquad \Gamma(x) = \sigma}{(\boldsymbol{\nu} x)\,\Gamma \vdash_{\mathcal{K}} (\boldsymbol{\nu} x : \sigma)\, P}\ \text{(E-Res)}$$

$$\frac{\begin{array}{c}\forall i : (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash_{\mathcal{K}} G_i.P_i \\ \Xi_{\mathrm{E}} \preceq \bigwedge_i \Xi_{\mathrm{E}i}\end{array}}{\left(\bigwedge_i \Sigma_i; \bigwedge_{k \in \mathcal{K}} \mathsf{sum}_k(\{p_i\}_i, \Xi_{\mathrm{E}}) \wedge \bigvee_i \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}}\right) \vdash_{\mathcal{K}} \sum_i G_i.P_i}\ \text{(E-Sum)}$$

$$\frac{\Gamma \vdash_{\mathcal{K}} P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x}}{\begin{array}{rl} \left(p : \sigma;\ \blacktriangleleft p^m \wedge \bar{p}^{m'}\right) & \odot \\ \left(; p^{\#(G)} \blacktriangleleft\right) & \odot \\ !_{\text{if } \#(G) = \omega}\ (\boldsymbol{\nu} \mathrm{bn}(G)) \Big(\ \Gamma \triangleleft \mathsf{dep}_{\mathcal{K}}(G) & \odot \\ \overline{\sigma}[\tilde{x}] \triangleleft (\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}}) & \odot \\ \left(; \bigwedge_{k \in \mathcal{K}} \mathsf{prop}_k(\sigma, G, m, m') \blacktriangleleft\right) & \Big) \vdash_{\mathcal{K}} G.P \end{array}}\ \text{(E-Pre)}$$

Table 5.1: Existential Type System Rules

**Definition 5.3.1 (Existential Type System)** Typability *of a typed process* $(\Gamma; P)$ *with respect to a set of universal and existential properties* $\mathcal{K}$ *including* $\mathbf{R}$*, written* $\Gamma \vdash_{\mathcal{K}} P$*, is inductively given by the rules in Table 5.1.*

A detailed typing example including explanations for the rules is given for Activeness in Section 6.4.

Definition 4.4.5 is augmented for existential properties, whose activeness become $\perp$ (unsatisfiable).

**Definition 5.3.2 (Binding)** *On dependencies,* $(\bar{\boldsymbol{\nu}} x)\varepsilon$ *is the logical homomorphism such that:*

$$(\bar{\boldsymbol{\nu}} x) p_k \overset{\text{def}}{=} \begin{cases} \perp & \text{if } \mathrm{n}(p) = x \text{ and } k \in \mathcal{E} \\ \top & \text{if } \mathrm{n}(p) = x \text{ and } k \in \mathcal{U} \\ p_k & \text{if } \mathrm{n}(p) \neq x \end{cases}$$

*On behavioural statements,* $(\boldsymbol{\nu} x)\,\Xi$ *is the logical homomorphism such that:*

$$(\boldsymbol{\nu} x)\,(p_k \triangleleft \varepsilon) = \begin{cases} \top & \text{if } \mathrm{n}(p) = x \\ p_k \triangleleft (\bar{\boldsymbol{\nu}} x)\varepsilon & \text{if } \mathrm{n}(p) \neq x \end{cases}$$

*On multiplicities:*

$$(\boldsymbol{\nu} x)\,(p^m) = \begin{cases} \top & \text{if } \mathrm{n}(p) = x \\ (p^m) & \text{if } \mathrm{n}(p) \neq x \end{cases}$$

*Binding a name* $x$ *in a process type* $\Gamma$ *is done as follows:*

$$(\boldsymbol{\nu} x)\ (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \overset{\text{def}}{=} \left(\Sigma|_{\mathrm{dom}(\Sigma) \setminus x}; (\boldsymbol{\nu} x)\,\Xi_{\mathrm{L}} \blacktriangleleft (\boldsymbol{\nu} x)\,\Xi_{\mathrm{E}}\right)$$

## 5.4 Events and Non-Transitive Dependencies

We describe in this section an extension to the typing notation that, although it isn't strictly necessary, significantly increases the set of processes correctly analysed by the type system. Namely, *events* permit non-transitive dependencies ($\alpha$ depending on $\beta$ and $\beta$ on $\gamma$ but $\alpha$ not depending on $\gamma$).

We use the *activeness* existential property, formally defined in Section 6, to motivate this extension. Consider the process $a(xy).\bar{x}.\bar{y} \,|\, \bar{a}\langle bc\rangle \,|\, b.c$, where all names are linear, active and responsive. It exhibits the following dependencies: By definition of responsiveness, $\bar{a}_{\mathbf{R}} \lhd (b_{\mathbf{A}} \wedge c_{\mathbf{A}})$. Because of prefixing, $c_{\mathbf{A}} \lhd \bar{b}_{\mathbf{A}}$. As $\bar{b}_{\mathbf{A}}$ is provided through parameter instantiation, it depends on $a$ being input active and responsive: $\bar{b}_{\mathbf{A}} \lhd a_{\mathbf{AR}}$. Collapsing these three dependencies would result in $\bar{a}_{\mathbf{R}} \lhd a_{\mathbf{R}}$, and a similar reasoning can be applied (just before the $(\boldsymbol{\nu} xy)$ binding done by the (E-PRE) rule) to show $a_{\mathbf{R}} \lhd \bar{a}_{\mathbf{R}}$, so we end up with $(a_{\mathbf{A}} \wedge \bar{a}_{\mathbf{R}}) \lhd \bot$. The problem is that output responsiveness should be computed *assuming the remote side is active and responsive* (available and behaving as specified in the channel type). So for the above example, when computing $\bar{a}_{\mathbf{R}}$'s dependencies, $\bar{b}_{\mathbf{A}}$ is assumed to be available. However, if $\bar{b}_{\mathbf{A}}$ is considered on its own, it does depend on both $a_{\mathbf{A}}$ and $a_{\mathbf{R}}$. Note that a common approach to this problem is to consider each parameter (and in turn their parameters, etc) as an individual resource (see e.g. Kobayashi) rather than grouping all of them into a single "responsiveness" resource.

A second example is $\bar{t}.(a(x).b.x \,|\, \bar{b})$, where $t$ is plain and $b$ linear. The $b.x$ part implies $a_{\mathbf{R}} \lhd \bar{b}_{\mathbf{A}}$. Because of prefixing, $\bar{b}_{\mathbf{A}} \lhd t_{\mathbf{A}}$. However, input responsiveness doesn't require the input to be available, but just that if it gets consumed, a reply will be sent. In this case, if the input is consumed then $\bar{t}$ must necessarily have been consumed as well, so that $b$ doesn't have dependencies. So $a_{\mathbf{R}} \lhd t_{\mathbf{A}}$ is *not* required, and we have $a_{\mathbf{R}} \lhd \top$.

For an (admittedly a bit far-fetched) example where $\bar{a}_{\mathbf{R}}$ appears at the other end of the chain, consider

$$q.\big(!\, z \mid \bar{a}\langle b\rangle \mid a(x).p(y).x.y\big) \mid \bar{p}\langle q\rangle \mid P$$

where $P$ contains active and responsive $a$-output and $p$-input. We have the chain $z_{\mathbf{A}} \lhd \bar{q}_{\mathbf{A}} \lhd p_{\mathbf{R}} \lhd \bar{a}_{\mathbf{R}} \lhd \bar{b}_{\mathbf{A}}$, but $z_{\mathbf{A}}$ does not depend on $\bar{b}_{\mathbf{A}}$, since by the time $\bar{a}\langle b\rangle$ comes to top-level, $z_{\mathbf{A}}$ no longer needs $\bar{q}_{\mathbf{A}}$, and so $\bar{q}_{\mathbf{A}}$'s dependency on $\bar{a}$'s responsiveness no longer matters. In other words, as long as the $q$-prefix hasn't been consumed, we have only $z_{\mathbf{A}} \lhd \bar{q}_{\mathbf{A}} \lhd p_{\mathbf{R}}$, and after $q$ has been consumed, we have $z_{\mathbf{A}}$ without dependencies and $\bar{q}_{\mathbf{A}} \lhd p_{\mathbf{R}} \lhd \bar{a}_{\mathbf{R}} \lhd \bar{b}_{\mathbf{A}}$.

We address all these cases through the concept of *events*. An event is a property related to the state of a process that either holds or doesn't. An example is "the $a$-server has received a query". Another example is "*this* and *that* prefixes have communicated" (where some unambiguous way to identify which prefixes "this" and "that" refer to is assumed).

The notation for process types from (1.2) on page 6 is extended as follows:

$$\Delta \ ::= \ \cdots \ \mid \ l \ \mid \ \bar{l} \tag{5.5}$$

We do not provide a way to formally express such an event, but only assume that, for a particular event and a particular state of a process, it has a well-defined truth value. Then, $l$ corresponds to $\top$ is the event has occurred, and to

$\bot$ if it has not. Its negation, $\bar{l}$, corresponds to $\bot$ if the event has occurred, and to $\top$ otherwise. To the definition of weakening we add the following rule:

$$l \vee \bar{l} \cong \top$$

In the first example above, let $l$ stand for "the communication on $a$ has taken place". Then responsiveness is vacuously true as long as $l$ did not occur, which can be expressed with $\bar{a}_{\mathbf{R}}\triangleleft(\bar{l} \vee (b_{\mathbf{A}} \wedge c_{\mathbf{A}}))$, and dependency on the remote activeness and responsiveness is only needed as long as $l$ has not taken place: $\bar{b}_{\mathbf{A}}\triangleleft(l \vee a_{\mathbf{AR}})$ and $\bar{c}_{\mathbf{A}}\triangleleft(b_{\mathbf{A}} \wedge (l \vee a_{\mathbf{AR}}))$. The rest stays the same: $b_{\mathbf{A}}$ and $c_{\mathbf{A}}\triangleleft\bar{b}_{\mathbf{A}}$. Substituting $b_{\mathbf{A}}$ by $\top$ and $c_{\mathbf{A}}$ by $\bar{b}_{\mathbf{A}}$ in the output responsiveness statement gives $\bar{a}_{\mathbf{R}}\triangleleft(\bar{l} \vee \bar{b}_{\mathbf{A}})$ as before. Substituting $\bar{b}_{\mathbf{A}}$ by $l \vee a_{\mathbf{AR}}$ yields $\bar{a}_{\mathbf{R}}\triangleleft(\bar{l} \vee l \vee a_{\mathbf{AR}})$ which is equivalent (by $l \vee \bar{l} \cong \top$ and $\top \vee \gamma \cong \top$) to $\bar{a}_{\mathbf{R}}\triangleleft\top$, i.e. $a$ is output responsive in the process.

As far as the second example is concerned, we have $a_{\mathbf{R}}\triangleleft(\bar{l}\vee\bar{b}_{\mathbf{A}})$ and $\bar{b}_{\mathbf{A}}\triangleleft(l \vee t_{\mathbf{A}})$, which combine into $a_{\mathbf{R}}\triangleleft(\bar{l} \vee l \vee t_{\mathbf{A}})$, which reduces to $a_{\mathbf{R}}\triangleleft\top$, as required.

When typing a process, *annotate* each guarded process $G.P$ with an event $l$ unique in the whole process (Section 7 explores thus *annotated processes* in more detail), as in $G^l.P$. The annotations can be discarded after the typing is done.

Dependencies of a guard $G^l$ extract the event tag ...

$$\mathsf{dep}_{\mathbf{A}}(G^l) \overset{\text{def}}{=} l \vee \overline{\mathsf{sub}(G)}_{\mathbf{A}}$$

... and so does the elementary responsiveness rule.

$$\mathsf{prop}_{\mathbf{R}}(\sigma, G^l, m, m') = \mathsf{sub}(G)_{\mathbf{R}}\triangleleft \begin{cases} \bar{l} \vee \sigma[\mathsf{obj}(G)] & \text{if } G \text{ is an input} \\ \bar{l} \vee \overline{\sigma}[\mathsf{obj}(G)] & \text{if } G \text{ is an output} \end{cases} \tag{5.6}$$

Finally, the $\bar{p}_{\mathbf{R}}$-dependency of remote behaviour in (E-PRE) must be similarly altered:

$$\frac{\Gamma \vdash_{\mathcal{K}} P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x}}{\begin{array}{r} \left(p : \sigma;\ \blacktriangleleft p^m \wedge \bar{p}^{m'}\right) \quad \odot \\ (; p^{\#(G)} \blacktriangleleft) \quad \odot \\ !_{\text{if } \#(G) = \omega}\ (\boldsymbol{\nu}\text{bn}(G)) \Big(\ \Gamma\triangleleft \mathsf{dep}_{\mathcal{K}}(G) \quad \odot \\ \overline{\sigma}[\tilde{x}]\triangleleft(\mathsf{dep}_{\mathcal{K}}(G) \wedge (l \vee \bar{p}_{\mathbf{R}})) \quad \odot \\ (; \bigwedge_{k\in\mathcal{K}} \mathsf{prop}_k(\sigma, G, m, m') \blacktriangleleft) \quad \Big) \vdash_{\mathcal{K}} G.P \end{array}} \text{(E-PRE)}$$

## 5.5 Delayed Dependencies and Self-Name Passing

Before summarising our results, we propose in this section another extension to the type notation that basically permits names passing references to themselves while still being responsive. We will not prove that these changes preserve the type system properties.

*Delayed dependencies* permit discarding certain circularities connecting two different depths of a recursive channel type, such as $!\,a(x).\overline{x}\langle a\rangle$ which is a server responding to queries by a pointer to itself. Another example is (10.1) on page 118 where $\mathsf{Geom}_{\mathbf{R}} \lhd \overline{succ}_{\mathbf{R}}$ and $\overline{succ}_{\mathbf{R}} \lhd \mathsf{Geom}_{\mathbf{R}}$ reduce to $\mathsf{Geom}_{\mathbf{R}} \lhd \top$ rather than $\mathsf{Geom}_{\mathbf{R}} \lhd \bot$. This extension can of course be applied simultaneously to the previous one since they operate on different parts of the theory.

In a statement $\gamma \lhd \varepsilon$, a resource $\alpha$ in $\varepsilon$ is now annotated with a *delay* $\alpha^d$ where $d$ is any number or $-\infty$ representing the "difference in depth" in the channel type. Note that, when this extension is in use, the abbreviation $(p^m_{\mathbf{A}} \lhd \varepsilon) = p^m \wedge (p_{\mathbf{A}} \lhd \varepsilon)$ should probably be avoided as it would create confusion.

The following examples illustrate nicely the semantics of delays.

- $a_{\mathbf{R}} \lhd {u_{\mathbf{A}}}^2 \ \vdash \ a(x).\overline{x}(\boldsymbol{\nu} y).\bar{u}.\bar{y}$ — the $u$-dependency is only required after *two* exchanges on $a$ (specifically, $a$ and $x$)

- $\bar{u}_{\mathbf{A}} \lhd {\bar{a}_{\mathbf{AR}}}^{-2} \ \vdash \ a(x).\overline{x}(\boldsymbol{\nu} y).\bar{u}.\bar{y}$ — the $\bar{u}$-output is available only after two exchanges on $a$ has been performed. Note the difference of the sign with the previous example ; $a_{\mathbf{R}}$ starts providing resources before needed $u_{\mathbf{A}}$, and $\bar{u}_{\mathbf{A}}$ needs $a_{\mathbf{AR}}$ before it provides resources.

- $b_{\mathbf{A}} \lhd {a_{\mathbf{A}}}^0 \ \vdash \ \bar{a}.b$ — the delay is 0 because $a_{\mathbf{A}}$ is needed before one can even start interacting with $b$.

- $a_{\mathbf{R}} \lhd {b_{\mathbf{AR}}}^0 \ \vdash \ !\, a(x).\bar{b}\langle x\rangle$ — a (depth 1) answer from $a$ depends on a (depth 1) answer from $b$.

- $a_{\mathbf{R}} \lhd {b_{\mathbf{AR}}}^2 \ \vdash \ !\, a(x).\overline{x}\langle b\rangle$ — in order to do $n$ steps of a dialogue with $a$, we need to be able to do $n-2$ steps of a dialogue with $b$.

When substituting resources for dependencies in the reduction relation "$\hookrightarrow$", $\varepsilon \mapsto \varepsilon^d$ is the logical homomorphism such that $\left(\alpha^d\right)^e = \alpha^{d+e}$ where $+$ is the usual numerical addition, extended with $\forall d : -\infty + d = -\infty$. When a substitution would introduce a self-dependency $\alpha \lhd \alpha^d$, $\alpha^d$ is replaced by $\top$ if $d > 0$, and $\bot$ otherwise.

Continuing the last example above, $!\, a(x).\overline{x}\langle b\rangle !\, b(x).\overline{x}\langle c\rangle$ would have type $a_{\mathbf{R}} \lhd {b_{\mathbf{AR}}}^2 \odot b_{\mathbf{R}} \lhd {c_{\mathbf{AR}}}^2$, which reduces to $a_{\mathbf{R}} \lhd {c_{\mathbf{AR}}}^{2+2} = a_{\mathbf{R}} \lhd {c_{\mathbf{AR}}}^4$. If $c = a$ then we get $a_{\mathbf{R}} \lhd \top$.

The channel instantiation operator $\sigma[\tilde{x}]$ adds the $-\infty$ delay to every dependency declared in the channel type, and circular dependencies added for completion have delay 0.

The transition operator delays responsiveness dependencies by $-1$:

$$\Gamma \wr a(\tilde{x}) \ \overset{\text{def}}{=} \ \Gamma \wr a \odot \sigma[\tilde{x}] \lhd \left({a_{\mathbf{R}}}^{-1} \blacktriangleleft {\bar{a}_{\mathbf{R}}}^{-1}\right)$$

and similarly for output, making explicit the fact that we descended one step into the channel type. For instance in $a(x).\bar{u}.\bar{x} \xrightarrow{a(t)} \bar{u}.\bar{t}$, the dependency $a_{\mathbf{R}} \lhd {u_{\mathbf{A}}}^1$ becomes $\bar{t}_{\mathbf{A}} \lhd {u_{\mathbf{A}}}^{1-1} = \bar{t}_{\mathbf{A}} \lhd {u_{\mathbf{A}}}^0$.

Finally, the prefix rule extended with delays is as follows:

$$\frac{\Gamma \vdash P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x}}{\begin{array}{rl} \Big(p : \sigma;\ \blacktriangleleft p^{m} \wedge \bar{p}^{m'}\Big) & \odot \\ \big(;\, p^{\#(G)} \blacktriangleleft\big) & \odot \\ !_{\text{if } \#(G) = \omega}\ (\boldsymbol{\nu}\mathrm{bn}(G)) \Big(\ \Gamma \triangleleft \mathsf{dep}_k(G)^0 & \odot \\ \overline{\sigma}[\tilde{x}] \triangleleft (\mathsf{dep}_k(G) \wedge \bar{p}_{\mathbf{R}})^{-1} & \odot \\ \big(;\, \bigwedge_{k \in \mathcal{K}} \mathsf{prop}_k(\sigma, G, m, m')^{+1} \blacktriangleleft\big) & \Big) \vdash_{\mathcal{K}} G.P \end{array}} \quad \text{(E-PRE)}$$

where $\Xi \mapsto \Xi^d$ is a logical homomorphism such that $(\alpha \triangleleft \varepsilon)^d \overset{\text{def}}{=} \alpha \triangleleft (\varepsilon^d)$.

## 5.6 Properties

This section summarises the properties enjoyed by the type system. It is a repeat of the corresponding results with just universal properties (Section 4.6).

**Lemma 5.6.1 (Decidability)** *Typability with with respect to a set of universal and existential properties is decidable.*

**Proposition 5.6.2 (Subject Congruence)** *Let* $\Gamma \vdash_{\mathcal{K}} P \equiv P'$*. Then* $\Gamma' \vdash_{\mathcal{K}} P'$ *for some* $\Gamma' \cong \Gamma$*.*

**Proposition 5.6.3 (Subject Reduction)** *Let* $(\Gamma; P)$ *be a typed process such that* $\Gamma \vdash_{\mathcal{K}} P$*. Then, for any transition* $(\Gamma; P) \xrightarrow{\mu} (\Gamma \wr \mu; P')$*,* $\exists \Gamma'$ *s.t.* $\Gamma' \preceq \Gamma \wr \mu$ *and* $\Gamma' \vdash_{\mathcal{K}} P'$*.*

The proof is in Appendix A.2.

**Proposition 5.6.4 (Type Soundness)** *If* $\Gamma \vdash_{\mathcal{K}} P$ *then* $\Gamma \models P$*.*

The proof is in Section 7.6.

# Chapter 6

# Activeness

## 6.1 Introduction

An common requirement one may wish to express about a component written in mobile calculus is that a process should be listening (respectively, ready to send) at an input (resp., output) port. Let's call this property *activeness* at a port[1]. Let's first review our needs before proceeding to a formal definition.

For example, consider a process decoding a value $v$ and sending a signal on a channel $s$: $P = a(v).\mathsf{case}\, v\, \mathsf{of}\, (x, y) : \bar{s}$, and a process first sending a signal and then decoding $v$: $Q = a(v).\bar{s}.\mathsf{case}\, v\, \mathsf{of}\, (x, y) : \mathbf{0}$. These processes could be encoded as $[\![ P ]\!] = a(u).\overline{u}(\boldsymbol{\nu} r_1 r_2).r_1(x).r_2(y).\bar{s}$ and $[\![ Q ]\!] = a(u).\bar{s}.\overline{u}(\boldsymbol{\nu} r_1 r_2).r_1(x).r_2(y).\mathbf{0}$, where $u$ holds an encoding of $v$.

As said in the introduction, $P \sim Q$ but $[\![ P ]\!] \not\approx [\![ Q ]\!]$ because they are distinguished by

$$R = \overline{a}(\boldsymbol{\nu} u).\bot.!\, u(xy).(\overline{x}\langle b\rangle | \overline{y}\langle c\rangle) \tag{6.1}$$

where

$$\bot.P \overset{\text{def}}{=} (\boldsymbol{\nu} t)\, t.P \tag{6.2}$$

with $t \notin \text{fn}(P)$. Note that $R$ does not violate any multiplicity constraint, as the receiver on $u$ is present — it is merely deadlocked (*inactive*).

Before I propose a solution, it should be noted that requiring $u$ to be active is not enough, as is shown by

$$R = \overline{a}(\boldsymbol{\nu} u).!\, u(xy).\bot.(\overline{x}\langle b\rangle | \overline{y}\langle c\rangle) \tag{6.3}$$

where $u$ is active (after the transition $\overline{a}(\boldsymbol{\nu} u)$), but, after $u$ receives a request $r_1 r_2$, the reply itself is not. This is solved by requiring *responsiveness* on $u$ (see Section 4.8) in addition to activeness.

Moreover, in order to have a property which is meaningful for nonlinear names, and for consistency with the liveness definition (Definition 5.2.2 on page 52), we add a *reliability* requirement to activeness.

Consider the process $P = p(x).\bar{x}$, where $p$ is plain (i.e. has multiplicities $p^\star \wedge \overline{p}^\star$). At first sight it might seem natural to declare that $p$ is active in $P$. However that input is not reliable because, composing $P$ with a process $\overline{p}\langle b\rangle.\bar{s}$ will not necessarily trigger the success signal $\bar{s}$, if a third party $E = \overline{p}\langle c\rangle$.

[1] *Input* activeness is commonly called receptiveness.

"steals" the input at $p$. In contrast, the replicated form $!\,P = !\,p(x).\bar{x}$ is reliable, because there is an infinite supply of inputs at $p$ and no third party can steal them all (assuming fairness on the scheduler).

Finally, our target being encodings, there will be typically an overhead (in terms of extra $\tau$-transitions) in an encoded process compared to the original one. Therefore it is acceptable if a number of $\tau$-transitions are required before a receiver (or sender, for output-activeness) becomes available. Ruling out such "weak activeness" would give *strong activeness* and is characterised by works such as [San99, ABL03].

This gives us an informal definition for activeness:

**Definition 6.1.1 (Activeness — Informal)** *A port $p$ is said* active *in a process $P$ if*

1. *$P$ will eventually (i.e. possibly after a finite number of $\tau$-reductions) contain an unguarded occurrence of $p$ in subject position.*
2. *The port is "reliable", in the sense that no third party can interfere in a way that prevents $p$ from being made available to a process attempting to communicate with that port.*

As the reader has no doubt guessed by now, the existential type system and liveness semantics can be instantiated to obtain precise activeness semantics and an associated sound type system.

We introduce the existential property $\mathbf{A}$. A resource $p_{\mathbf{A}}$ in behavioural statements, meaning that the port must be used at least once. Note that multiplicities and activeness are complementary, in that the former put an upper bound to the number of uses of a channel, and the latter puts a *lower* bound on that number.

Assuming $\sigma_p$ is the type for $b$ and $c$ in the example at the beginning of this section, the reply channels $r_1$ and $r_2$ will have a type such as

$$\sigma_r = (\sigma_p; 1^\star \wedge \bar{1}^\star; 1^\star \wedge \bar{1}^\star)$$

The $\star$ exponents and absence of $\mathbf{A}$-resources mean that both the input and output ports of reply channels are free to interact with the parameters $b$ and $c$ in any way. A type for $u$ can then be written $\sigma_u = (\sigma_r, \sigma_r; \bar{1}_{\mathbf{A}} \wedge \bar{2}_{\mathbf{A}}; 1_{\mathbf{A}} \wedge 2_{\mathbf{A}})$, telling that $u$'s input port must provide one active output on both parameters, and $u$'s output port must provide one active input on both parameters. Finally, the channel $a$ will have a type such as $(\sigma_u; \bar{1}^\star; 1^\omega_{\mathbf{A}} \wedge \bar{1}^\star)$, where both input and output ports of $a$ may send requests on the parameter $u$ but $a$'s *output* port must provide one replicated ("$\omega$") and active ("$\mathbf{A}$") input at the parameter.

Note that it makes little sense to specify activeness on the environment component of a process type, so we will usually have activeness marks on the local component only.

Some examples:
The type $\big(a : (), b : (); a_{\mathbf{A}} \wedge b \blacktriangleleft \bar{a} \wedge \bar{b}\big)$ is a valid description of $a\,|\,b$, of $a.b$ and $a\,|\,\bot.b$, but not of $\bot.a\,|\,b$. It does however correctly describe

$$\tau.a \overset{\text{def}}{=} (\boldsymbol{\nu} t)\,(\bar{t}|t.a.\mathbf{0})$$

as the fact that $a$ is not immediately available is not an issue if it is guaranteed to eventually become so.

The type $\big(a : (); a^{\star}_{\mathbf{A}} \blacktriangleleft a^0 \wedge \bar{a}^{\star}\big)$ is a valid description of $!\, a.\mathbf{0}$, but not of $a.\mathbf{0}$, because the latter is unreliable. $\big(a : (); a^{\star}_{\mathbf{A}} \blacktriangleleft a^0 \wedge \bar{a}^{1}\big)$, on the other hand, is a valid description of both processes: As the environment may only do one output on $a$, there is no risk of competition even if the input is not replicated.

Finally, using the notation

$$?.P \stackrel{\text{def}}{=} (\boldsymbol{\nu} t)\,(\bar{t} \,|\, t \,|\, t.P) \tag{6.4}$$

($t$ fresh) as a shortcut for an "unreliable prefix", $\big(a : ((); \bar{1}_{\mathbf{A}}; 1_{\mathbf{A}}); a_{\mathbf{A}} \blacktriangleleft a^0 \wedge \bar{a}\big)$ is a valid description of $a(x).\bar{x}$, but neither describes $?.a(x).\bar{x}$ ($a$ is not active) nor $a(x).?.\bar{x}$ ($x$ is not active).

Weakening the process type to $\big(a : ((); \bar{1}_{\mathbf{A}}; 1_{\mathbf{A}}); a \blacktriangleleft a^0 \wedge \bar{a}\big)$ allows describing the first two processes, but still not the last: It is no longer required for $a$ to be active, but if a request is received then it *must* be replied to, because the parameter is declared active in the channel type.

The input port of a Boolean channel (such as $r$, $a$ and $b$ in (1.1), page 5) has type

$$\bar{1}^1_{\mathbf{A}} \;\vee\; \bar{2}^1_{\mathbf{A}}, \tag{6.5}$$

that says that either the first parameter ("1") must be output ("$\bar{1}$") active ("$_{\mathbf{A}}$"), and the second parameter unused[2], or ("$\vee$") the opposite.

The Boolean protocol requires outputs to provide a *branching* on the parameters, so for instance

$$\bar{b}(\boldsymbol{\nu} tf).(t.P{+}f.Q) \tag{6.6}$$

is a responsive client (correctly implementing "if $b$ then $P$ else $Q$"), while, defining the internal choice operator $\oplus$ as

$$P \oplus Q \stackrel{\text{def}}{=} (\boldsymbol{\nu} t)\,\big(\bar{t} \,|\, (t.P{+}t.Q)\big) \tag{6.7}$$

for some $t \notin (\mathrm{fn}(P) \cup \mathrm{fn}(Q))$,

$$\bar{b}(\boldsymbol{\nu} tf).(t.P \oplus f.Q) \tag{6.8}$$

may lead to deadlocks. We want the first process to be recognised as correct and the second one to be ruled out but of course both obey the client protocol $1^1_{\mathbf{A}} \vee 2^1_{\mathbf{A}}$. We need a way to have behavioural statements express the property "1 and 2 must be guards of a sum".

To this end we extend the grammar for resources:

$$\alpha \;::=\; p_k \;\big|\; s_{\mathbf{A}} \tag{6.9}$$

$$s \;::=\; p \;\big|\; (p+s) \tag{6.10}$$

Just like $p_{\mathbf{A}}$, activeness of a port $p$, requires a $p$-guarded process to eventually come to top-level, activeness of a branching $(\sum_i p_i)_{\mathbf{A}}$ requires a sum to eventually come to top-level, with one $p_i$-guarded branch for each $i$.

We can now write the output Boolean protocol:

$$\big(1^1 \;\vee\; 2^1\big) \wedge (1+2)_{\mathbf{A}}, \tag{6.11}$$

[2]Remember Convention 4.2.2 on page 32: ports that aren't mentioned have multiplicity zero.

which is similar to (6.5) but on the input port of the parameters, and with the additional constraint ("$\wedge$") that inputs at the parameters ("1" and "2") must be the guards of a sum ("+"). This protocol is respected by (6.6) and broken by (6.8).

Abbreviating the parameter-less channel type $(;;)$ as $()$, the Boolean type gathers (6.5) and (6.11) as

$$\mathsf{Bool} \overset{\text{def}}{=} \big(()() \ ; \ \bar{1}^1_{\mathbf{A}} \ \vee \ \bar{2}^1_{\mathbf{A}} \ ; \ (1^1 \vee 2^1) \wedge (1+2)_{\mathbf{A}}\big) \tag{6.12}$$

Consider the following process:

$$t.a(x).u.\bar{x}$$

As far as activeness is concerned, we have $t_{\mathbf{A}} \lhd \top$, $a_{\mathbf{A}} \lhd \bar{t}_{\mathbf{A}}$, $u_{\mathbf{A}} \lhd (\bar{t}_{\mathbf{A}} \wedge \bar{a}_{\mathbf{A}})$, and, after $a$ has been consumed and $x$ made visible, $\bar{x}_{\mathbf{A}} \lhd \bar{u}_{\mathbf{A}}$.

By definition, $a_{\mathbf{R}} \lhd \bar{x}_{\mathbf{A}}$ ($a$ is responsive if $\bar{x}$ is active), which gives us $a_{\mathbf{R}} \lhd \bar{u}_{\mathbf{A}}$. Why doesn't $a$'s responsiveness depend on $\bar{t}_{\mathbf{A}}$? The idea is that responsiveness's dependencies are those that are required to provide a reply *after a request has been received.* In this case, $\bar{t}_{\mathbf{A}}$ is no longer needed once $a$ has received a request, but $\bar{u}_{\mathbf{A}}$ is required to answer it. Inversely, $\bar{t}_{\mathbf{A}}$ is required for a communication on $a$ to take place, but $\bar{u}_{\mathbf{A}}$ is not needed for that.

The following process (where $a$ is plain active) is another illustration of the duality between activeness and responsiveness:

$$t_1.a(x).u_1.\bar{x} \mid t_2.a(x).u_2.\bar{x}$$

Now we have $a_{\mathbf{A}} \lhd (\bar{t}_{1\mathbf{A}} \vee \bar{t}_{2\mathbf{A}})$ and $a_{\mathbf{R}} \lhd (\bar{u}_{1\mathbf{A}} \wedge \bar{u}_{2\mathbf{A}})$: any of the $\bar{t}_{i\mathbf{A}}$ must be provided for $a$ to be active, but *both* $\bar{u}_{i\mathbf{A}}$ must be provided for $a$ to be responsive. The reason is that the sender can't know for certain which input on $a$ will receive the request, and therefore must provide both $\bar{u}_i$ to be certain the request gets replied.

The following process shows why keeping activeness and responsiveness separate when computing dependencies is interesting:

$$\bar{a}\langle t\rangle.\mathbf{!}\, b(x).\bar{x} \mid \mathbf{!}\, a(y).\bar{b}\langle y\rangle \tag{6.13}$$

We have both $b_{\mathbf{A}} \lhd a_{\mathbf{A}}$ (because of the left-hand component) and $a_{\mathbf{R}} \lhd b_{\mathbf{R}}$ (because of the right-hand component), and yet the process isn't deadlocked. However, not distinguishing $a_{\mathbf{A}}$ and $a_{\mathbf{R}}$ would result in the circularity "$a \lhd b \lhd a$" and have the process rejected.

We can now add activeness annotation to (3.2) as a type for the process (1.1). The local behavioural statement states that $r$ is active with multiplicity $\omega$ (i.e. has precisely one occurrence and it is replicated), and its responsiveness depends on both $a$ and $b$ being active and responsive. The environment component specifies that $a$ and $b$ must both have at most one replicated instance.

$$\begin{aligned}\Gamma_A = \big(&a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}; \\ &\big(r^{\omega} \wedge (r_{\mathbf{A}} \lhd \top)\big) \wedge \big(r_{\mathbf{R}} \lhd (a_{\mathbf{A}} \wedge a_{\mathbf{R}} \wedge b_{\mathbf{A}} \wedge b_{\mathbf{R}})\big) \blacktriangleleft a^{\omega} \wedge b^{\omega}\big)\end{aligned} \tag{6.14}$$

By convention 4.2.2, the previous type can be rewritten:
$(a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}; r^{\omega}_{\mathbf{A}} \wedge r_{\mathbf{R}} \lhd (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \blacktriangleleft a^{\omega} \wedge b^{\omega})$

## 6.2 Branching Algebra

As explained earlier, activeness is a property of a set of ports rather than a single port. This means that, while all laws in Section 5 refer to resources using the notation "$p_k$", the $p$ can now also be a *branching* $s = p_1 + p_2 + \cdots p_n$. The previously introduced rules are otherwise unchanged. We now introduce rules dealing specifically with sums.

Removal of non-observable dependencies needs to check observability of all ports in a branching. Note that the check for $p_i \neq p_j$, as well as the condition on "at most one" $p_i$ being observable are only required because we're overloading the **A** property for both sum activeness and port activeness. Also note that in $\Gamma = \big(\Sigma; t \vee f \blacktriangleleft \bar{t} \vee \bar{f}\big)$, a resource $(t+f)_{\mathbf{A}}$ would be preserved, as both $t$ and $f$ are observable as $\Gamma \wr t$ and $\Gamma \wr f$ are both well defined and equal to $(\Sigma; \top \blacktriangleleft \top)$ — they just aren't observable *simultaneously.*

**Definition 6.2.1 (Removal of Non-Observable Dependencies)** *Let* $\Gamma$ *be a process type.* Removing *non-observable dependencies from it is done as in Definition 5.1.5, with the following additional rules as well, where* $s_{\mathbf{A}}$ *ranges over branching resources of the form* $\big(\sum_{i\in I} p_i\big)_{\mathbf{A}}$ *and there are* $i$, $j$ *with* $p_i \neq p_j$.

- *Replace any statement* $s_{\mathbf{A}} \lhd \varepsilon$ *where at most one* $p_i$ *is observable in* $\Gamma$ *by* $\top$
- *In any statement* $\gamma \lhd \varepsilon$, *replace any* $s_{\mathbf{A}}$ *in* $\varepsilon$ *by* $\bot$ *if at least one* $p_i$ *isn't observable.*

The $p$-reduction operator (Definition 3.7.1 on page 24) works similarly on branching resources: $\big(\sum_i p_i\big)_{\mathbf{A}} \wr p \stackrel{\text{def}}{=} \top$ if both $p = p_i$ and $p \neq p_{i'}$ for some $i \neq i'$.

To the dependency reduction operator $\hookrightarrow$ (Definition 5.1.3 on page 49) we add the following rule:

For $m \neq 0$, $p \neq q$ and $\varepsilon \ncong \bot \ncong \varepsilon'$:

$$(p+q+s)_{\mathbf{A}} \lhd \varepsilon \;\wedge\; p_{\mathbf{A}} \lhd \varepsilon' \;\wedge\; \bar{q}^m \hookrightarrow \bot$$

This rule simulates a selection and a branching occurring inside a process, by replacing every term of the branching that does not match the selection by $\bot$, which is the neutral element of $\vee$. For example the transition $\bar{t} \mid (t.P{+}f.Q) \xrightarrow{\tau} P$ is matched by $(t{+}f)_{\mathbf{A}} \wedge \big((t_{\mathbf{A}} \wedge \Gamma_P) \vee (f_{\mathbf{A}} \wedge \Gamma_Q)\big) \wedge \bar{t}^1 \cong \big((t{+}f)_{\mathbf{A}} \wedge t_{\mathbf{A}} \wedge \Gamma_P \wedge \bar{t}^1\big) \vee \big((t{+}f)_{\mathbf{A}} \wedge f_{\mathbf{A}} \wedge \Gamma_Q \wedge \bar{t}^1\big) \hookrightarrow \big((t{+}f)_{\mathbf{A}} \wedge t_{\mathbf{A}} \wedge \Gamma_P \wedge \bar{t}^1\big) \vee \bot \cong \big((t{+}f)_{\mathbf{A}} \wedge t_{\mathbf{A}} \wedge \Gamma_P \wedge \bar{t}^1\big)$. We require activeness of the branching to prevent the rule from applying in case there is a risk of race conditions.

The binding operator $(\boldsymbol{\nu} x)$ (Definition 5.3.2 on page 56) works on sums as follows:

$$(\boldsymbol{\nu} x)\left(\left(\sum_{i\in I} p_i\right)_{\mathbf{A}} \lhd \varepsilon\right) = \left(\sum_{i\in I : \mathrm{n}(p_i)\neq x} p_i\right)_{\mathbf{A}} \lhd (\bar{\boldsymbol{\nu}} x)\varepsilon$$

The degenerated case where $\{i \in I : \mathrm{n}(p_i) \neq x\}$ is empty gives just $\top$. Also note how, when $I$ contains a single element, this rule reduces to the one in Definition 5.3.2.

We illustrate the transition operator on the process $A$ (Equation 1.1):

The transition $A \xrightarrow{r(uv)} A' = A \,|\, \overline{a}(\boldsymbol{\nu} t'f').(t'.\overline{b}\langle uv\rangle + f'.\overline{v})$ is matched on $\Gamma_A$ (6.14) by

$$\big(\Sigma; r^{\omega}_{\mathbf{A}} \wedge r_{\mathbf{R}} \lhd (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \blacktriangleleft a^{\omega} \wedge b^{\omega} \wedge r^{0}\big) \wr r(uv) = \\ \Gamma_A \wr r \odot (u : (), v : (); (\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd r_{\mathbf{R}} \blacktriangleleft (u+v)_{\mathbf{A}} \lhd \bar{r}_{\mathbf{R}} \wedge (u \vee v))$$

The "$\wr r$" part has no effect, as discussed when illustrating Definition 3.7.1. Computing the composition works as follows, where the numbers match those in Definition 4.2.6.

1. The channel type mapping of the resulting process type is just $a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}, u : (), v : ()$. The local component is

   $$r^{\omega}_{\mathbf{A}} \wedge r_{\mathbf{R}} \lhd (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \odot (\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd r_{\mathbf{R}} = \\ r^{\omega}_{\mathbf{A}} \wedge r_{\mathbf{R}} \lhd (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \wedge (\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd r_{\mathbf{R}}$$

   and the environment component is just the conjunction $\big(a^{\omega} \wedge b^{\omega} \wedge r^{0}\big) \wedge \big((u+v)_{\mathbf{A}} \lhd \bar{r}_{\mathbf{R}} \wedge (u \vee v)\big)$.
2. Closure of the resulting expression reduces the $(\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd r_{\mathbf{R}} \;\wedge\; r_{\mathbf{R}} \lhd (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}})$ dependency chain, producing the statement $(\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd (r_{\mathbf{R}} \wedge a_{\mathbf{AR}} \wedge b_{\mathbf{AR}})$.
3. Finally, because of $r^0$ in the environment component, the dependency on $r_{\mathbf{R}}$ can be replaced by $\top$ in the above statement, resulting in $(\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}})$.

Omitting the parts about $r$'s activeness and responsiveness that were left unchanged, we end up with

$$\big(a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}, u : (), v : (); \\ (\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \blacktriangleleft \\ a^{\omega} \wedge b^{\omega} \wedge r^{0} \wedge (u \vee v)\big) \quad (6.15)$$

as a type for $A \,|\, \overline{a}(\boldsymbol{\nu} t'f').(t'.\overline{b}\langle uv\rangle + f'.\overline{v})$, where the local behavioural statement is read as "if active and responsive $a$ and $b$ inputs are provided, then an output will be sent on (exactly) one of $u$ and $v$," which is indeed a correct statement for that process $A'$.

Remember that this type was not obtained by analysing $A'$, but is a prediction of the effect of a transition $\xrightarrow{r(uv)}$ on a process of type $\Gamma_A$.

## 6.3 Activeness Semantics

In this section we give semantic definitions for the liveness property "**A**", as an instance of liveness semantics (Section 5.2).

The $\mathsf{good}_{\mathbf{A}}$ predicate (characterising immediate correctness of an activeness statement) is precisely what we called *strong activeness* in Section 6.1 above.

**Definition 6.3.1 (Immediate Correctness)** *An atomic statement* $s_{\mathbf{A}} \lhd \varepsilon$ *is* immediately correct *in a typed process* $(\Gamma; P)$ *(written* $\mathsf{good}_{\mathbf{A}}(s \lhd \varepsilon, (\Gamma; P))$*) if it satisfies one of the following rules.*

- *A behavioural statement $s_{\mathbf{A}} \lhd \bot$ is always immediately correct.*
- *An activeness statement $(\sum_{i \in I} p_i)_{\mathbf{A}} \lhd \varepsilon$ is immediately correct if $P \equiv (\boldsymbol{\nu}\tilde{z})\big((\sum_{j \in J} G_j.C_j) \mid Q\big)$ with $I \subseteq J$ and $\forall i \in I : \mathsf{sub}(G_i) = p_i$ and $\mathrm{n}(p_i) \notin \tilde{z}$.*

The correctness Definition (5.2.6) — working on port-based properties $p_k$ — works precisely the same way on $s_{\mathbf{A}}$ which is a property of a set of ports, you just need to replace $p$ by $s$ in the Definition.

A strategy doing a labelled transition depends on activeness of the complement port (see Definition 5.2.5).

**Definition 6.3.2 (Activeness Transition Dependencies)**
*The activeness dependencies of transition $\mu$ are given by $\mathsf{dep}_{\mathbf{A}}(\mu) = \overline{\mathsf{sub}(\mu)}_{\mathbf{A}}$.*

To conclude the semantics part, let's sketch a proof that $\Gamma_A$ given in (6.14) is a correct type for $A$ given in (1.1). We only pick a representative transition sequence, but of course a complete proof would have to take all possible transitions into account.

Following the pattern given in Definition 5.2.6 we shall alternate arbitrary transition sequences $\tilde{\mu}_i$ (odd-numbered steps below) and transitions provided by the strategy (even-numbered steps below).

1. We start by sending a request $\tilde{\mu}_0 = r(uv)$ to the process. The resulting type is given in (6.15), and its behavioural statement is already elementary ($\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{R}}$ contains no "$\wedge$" and $a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}$ contains no "$\vee$")

2. To bring the process closer to an output on $u$ or $v$, the strategy sends the $\bar{a}(\boldsymbol{\nu}t'f')$ output, which is permitted because its subject $\bar{a}$ has its complement $a$ active in the dependencies. The local behavioural statement is now

$$(\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd (a_{\mathbf{AR}} \wedge (\bar{t}'_{\mathbf{A}} \vee \bar{f}'_{\mathbf{A}}) \wedge b_{\mathbf{AR}})$$

3. As we do not want to help the strategy find the way out we set $\tilde{\mu}_1 = \varnothing$. However we must still do a projection "$\searrow$" which is not trivial because now the dependency contains a disjunction. In other word we must simulate the choice made by the $a$ input. Let's pick $\bar{f}'$:

$$(\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \lhd (a_{\mathbf{AR}} \wedge \bar{f}'_{\mathbf{A}} \wedge b_{\mathbf{AR}})$$

4. The process is now
$$A \mid (t'.\bar{b}\langle uv\rangle + f'.\bar{v})$$
so the strategy is just to consume the $f'$ prefix, which is permitted because its complement is active ($\bar{f}'_{\mathbf{A}}$) in the dependencies.

5. We are now at the process $A|\bar{v}$. If we do nothing at this point ($\tilde{\mu}_2 = \varnothing$), $n = 2$ satisfies the requirement as $\bar{v}_{\mathbf{A}}$ is now immediately correct. If instead we consume $\bar{v}$ with $\tilde{\mu}_2 = \bar{v}$, the transition operator removes activeness of both $\bar{u}$ and $\bar{v}$ (see Definition 5.1.3 and the discussion that follows it). in other words, it replaces the dependency on $(a_{\mathbf{AR}} \wedge \bar{f}'_{\mathbf{A}} \wedge b_{\mathbf{AR}})$ by a dependency on $\bot$ which, by the first point of Definition 6.3.1, is always immediately correct.

Picking $\bar{t}'$ instead of $\bar{f}'$ at step 3 is essentially the same: the strategy then follows the $\xrightarrow{t'}\xrightarrow{\bar{b}\langle uv\rangle}$ path and the transition operator drops $\bar{u} \vee \bar{v}$ at the second transition.

We provide the definition of the $\mathsf{prop}_{\mathbf{A}}$ operator in order to instantiate the type system given in Section 5.3:

**Definition 6.3.3 (Activeness Guard Rule)**

$$\mathsf{prop}_{\mathbf{A}}(G, \sigma, m, m') = \begin{cases} \mathsf{sub}(G)_{\mathbf{A}} & \textit{if } \#(G) = \omega \textit{ or } m' \neq \star \\ \top & \textit{otherwise} \end{cases}$$

The activeness sum elementary rule is responsible for introducing sum activeness.

Activeness of a branching is guaranteed by a process type having no "concurrent environment $p_i$", making sure that any attempt to select a branch of such a sum (by communicating with its guard) will succeed.

**Definition 6.3.4 (Concurrent Port Use)** *Let $\{p_i\}_{i \in I}$ be a set of ports.*

- *A behavioural statement $\Xi$ is said to have* concurrent $p_i$ *if $\nexists i \in I$ such that $\bigwedge_{i' \in I \setminus i}({p_{i'}}^0) \preceq \Xi$.*

- *A behavioural statement $\Xi \vee \Xi'$ has concurrent $p_i$ if and only if (at least) one of $\Xi$ or $\Xi'$ has.*

- *A process type $(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ has concurrent environment $p_i$ if and only if $\Xi_{\mathrm{E}}$ has concurrent $p_i$.*

**Definition 6.3.5 (Activeness Sum Rule)**

$$\mathsf{sum}_{\mathbf{A}}(\{p_i\}_i, \Xi) = \begin{cases} \top & \textit{if } \Xi \textit{ has concurrent environment } p_i \\ (\sum_i p_i)_{\mathbf{A}} & \textit{otherwise} \end{cases}$$

To obtain the $\Xi_{\mathrm{E}}$ term in (E-SUM), the simplest way is to just leave $\Xi_{\mathrm{E}}$ at the weakest possible permitted by the rule, but this is usually not desirable because it often causes the sum activeness to drop. On the other hand this permits deactivating the type system check for race-conditions like

$$a.P{+}b.Q \mid \bar{a} \mid \bar{b} \tag{6.16}$$

A usually preferable way is to take

$$\Xi_{\mathrm{E}} = \bigwedge_i \Xi_{\mathrm{E}i} \wedge \bigvee_i \bigwedge_{j \neq i} \bar{p}_j^0$$

which forces $(\sum_i p_i)_{\mathbf{A}}$ to hold, but would reject (6.16) as unsafe.

## 6.4 A Typing Example

We now illustrate the type system by proving that

$$r_{\mathbf{R}} \triangleleft (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \tag{6.17}$$

($r$ is responsive if both $a$ and $b$ are active and responsive) can be built from the process (1.1) on page 5. All rules of the type system except (E-PAR) are used in this derivation so we'll describe them in the order they are used. For an explanation of (E-PAR), refer to the description of $\odot$ in Section 3.9. The reader may want to follow the rules on page 56 in parallel with this development.

Strictly following the rules gives a behavioural statement containing every possible statement that can be made about the process, so types can become rather large even for simple processes. So in this example we omit parts of the types that are not used to compute $r$'s responsiveness dependencies. Typing is syntax directed, starting from invocations of (E-NIL) (that types the idle process with the neutral element of $\odot$).

We start with the parameter-less output $\bar{f}$, which is typed using the prefix rule (E-PRE). The name is linear ($m = m' = 1$) and, since there are no parameters or continuation, only the first two factors of the typing, as well as the $k = \mathbf{A}$ in the last one, are non-empty (different from $\odot$'s neutral element), leaving us with: $\big(f : (); \blacktriangleleft \bar{f}^1 \wedge f^1\big) \odot \big(; \bar{f}^1 \blacktriangleleft\big) \odot (; \mathsf{prop}_{\mathbf{A}}(\sigma, G, m, m') \blacktriangleleft)$, that is:

$$\big(f : (); \bar{f}_{\mathbf{A}} \blacktriangleleft \bar{f}^0 \wedge f^1\big) \vdash \bar{f} \tag{6.18}$$

Sequence $G.P$ is typed much like parallel composition $G|P$, except that existential resources in $P$ additionally depend on $\mathsf{dep}_{\mathbf{A}}(G) = \overline{\mathsf{sub}(G)}_{\mathbf{A}}$, activeness of the complement of $G$'s subject port $\mathsf{sub}(G)$. Thanks to this, analysing a bound output $\overline{a}(\boldsymbol{\nu} b).P_b$ (where $P_b$ is an input on $b$) or its encoding $(\boldsymbol{\nu} b)\,(\overline{a}\langle b\rangle \,|\, P_b)$ in asynchronous $\pi$-calculus produces the exact same type. For our process, $f'.\bar{f}$ is again typed with (E-PRE), where all terms but the fourth are now non-null and $\Gamma$ is the type of the continuation given in (6.18):

$$\Big(f' : (); \blacktriangleleft f'^1 \wedge \bar{f}'^1\Big) \odot \Big(; f'^1_{\mathbf{A}} \triangleleft \top \blacktriangleleft\Big) \odot \Gamma \triangleleft \bar{f}'_{\mathbf{A}} \vdash_{\mathbf{AR}} f'.\bar{f}$$

Dropping the $f'^1_{\mathbf{A}}$ statement we get

$$\Big(f : (), f' : (); \bar{f}_{\mathbf{A}} \triangleleft \bar{f}'_{\mathbf{A}} \blacktriangleleft \bar{f}^0 \wedge f^1 \wedge f'^0 \wedge \bar{f}'^1\Big) \vdash_{\mathbf{AR}} f'.\bar{f} \tag{6.19}$$

Remote behaviour $\overline{\sigma}[tf] \triangleleft \bar{p}_{\mathbf{AR}}$ states that, if the input on $b$ is active and responsive then it will behave according to the protocol specified in the channel type whenever queries are sent to it. For the $\bar{b}\langle tf\rangle$ process, this is written $(\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \triangleleft b_{\mathbf{AR}}$, where the left side is just (6.11) from page 63 with $t$ and $f$ replacing 1 and 2 (and omitting terms with a zero exponent). The environment component $t^1 \vee f^1$ limits many times the local side is permitted to use the parameters' ports, which effectively prevents any part of the process to do at $t$ and $f$ anything more than a input-guarded sum at $t$ and at $f$. Together with the subject $b$ handled as in previous examples, we get the following:

$$\big(b : \mathsf{Bool}, t : (), f : (); (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \triangleleft b_{\mathbf{AR}} \blacktriangleleft (t^1 \vee f^1) \wedge (\bar{b}^\star \wedge b^\omega)\big) \vdash_{\mathbf{AR}} \bar{b}\langle tf\rangle \tag{6.20}$$

As in (6.19), the $t'$-prefix adds a dependency on $\mathsf{dep}_{\mathbf{A}}(t') = \bar{t'}_{\mathbf{A}}$ to all activeness resources, effectively turning the $b_{\mathbf{AR}}$ dependency into $b_{\mathbf{AR}} \wedge \bar{t'}_{\mathbf{A}}$:

$$\big(\Sigma; (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \lhd (b_{\mathbf{AR}} \wedge \bar{t'}_{\mathbf{A}}) \blacktriangleleft (t^1 \vee f^1) \wedge (\bar{b}^\star \wedge b^\omega)\big) \vdash_{\mathbf{AR}} t'.\bar{b}\langle tf\rangle \tag{6.21}$$

A sum $T+F$ is given, through (E-Sum), the type $(t'+f')_{\mathbf{A}} \lhd \varepsilon \wedge (\Gamma_T \vee \Gamma_F)$, where $\Gamma_T$ and $\Gamma_F$ are respectively the types of $T$ and $F$, and $t'$, $f'$ their guards: depending on the above definition, the process may ($\varepsilon = \top$) or may not ($\varepsilon = \bot$) offer a branching $t'+f'$, and, in addition ("$\wedge$") selects ("$\vee$") one of $\Gamma_T$ and $\Gamma_F$. The decoupling between the guards and the continuations is done to make explicit which channels must be used to make the process branch. Note how the original existential type system, without support for sum activeness, does not do this distinction and therefore gives precisely the same type to $P+Q$ and $P \oplus Q$.

In the example (1.1), in addition to $(t'+f')_{\mathbf{A}}$, the type for the continuation of the $a$-output is obtained from (6.19) and (6.21):

$$\begin{aligned}\big(\Sigma; (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \lhd (b_{\mathbf{AR}} \wedge \bar{t}'_{\mathbf{A}}) \vee (\bar{f}_{\mathbf{A}} \lhd \bar{f}'_{\mathbf{A}}) \blacktriangleleft \\ (t^1 \vee f^1) \wedge \bar{b}^\star \wedge b^\omega \wedge \bar{f}^0 \wedge {f'}^0 \wedge \bar{f'}^1\big) \vdash_{\mathbf{AR}} t'.\bar{b}\langle tf\rangle + f'.\bar{f}\end{aligned} \tag{6.22}$$

We run (E-Pre) once more in order to type the full $a$-output. Now the guard has two bound names $\mathrm{bn}(\bar{a}(\boldsymbol{\nu} t'f')) = \{t', f'\}$. For our purposes we only need the third and fourth terms:

- Remote behaviour $\Big(t' \colon (), f' \colon (); (\bar{t}'_{\mathbf{A}} \vee \bar{f}'_{\mathbf{A}}) \lhd a_{\mathbf{AR}} \blacktriangleleft {t'}^1 \vee {f'}^1\Big)$
- Continuation

$$\begin{aligned}\big(\Sigma; (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \lhd (b_{\mathbf{AR}} \wedge \bar{t}'_{\mathbf{A}} \wedge a_{\mathbf{A}}) \vee (\bar{f}_{\mathbf{A}} \lhd (\bar{f}'_{\mathbf{A}} \wedge a_{\mathbf{A}})) \blacktriangleleft \\ (t^1 \vee f^1) \wedge \bar{b}^\star \wedge b^\omega \wedge \bar{f}^0 \wedge {f'}^0 \wedge \bar{f'}^1\big)\end{aligned}$$

For the first time, the $\odot$ operator has to do dependency reduction (Definition 5.1.3 on page 49): The remote behaviour provides either $\bar{t}'_{\mathbf{A}} \lhd a_{\mathbf{AR}}$ or $\bar{f}'_{\mathbf{A}} \lhd a_{\mathbf{AR}}$, and in the continuation either $(\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}})$ depends on $\bar{t}'_{\mathbf{A}}$, or $\bar{f}_{\mathbf{A}}$ depends on $\bar{f}'_{\mathbf{A}}$. Remember, for existential resources like activeness, if $\alpha$ depends on $\beta$ and $\beta$ on $\gamma$, then $\alpha$ depends on $(\beta \vee \gamma)$, so the two behavioural statements in the continuation become respectively $(\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \lhd (\bar{t}'_{\mathbf{A}} \vee a_{\mathbf{AR}})$ and $\bar{f}_{\mathbf{A}} \lhd (\bar{f}'_{\mathbf{A}} \vee a_{\mathbf{AR}})$.

Combining the above five factors and binding $t'$ and $f'$ yields the following:

$$\begin{aligned}\big(a : \mathsf{Bool}, t : (), f : (); \\ (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \lhd (b_{\mathbf{AR}} \wedge a_{\mathbf{AR}}) \vee \bar{f}_{\mathbf{A}} \lhd a_{\mathbf{AR}} \blacktriangleleft \\ a^\omega \wedge (t^1 \vee f^1)\big) \vdash_{\mathbf{AR}} \bar{a}(\boldsymbol{\nu} t'f').(t'.\bar{b}\langle tf\rangle + f'.\bar{f})\end{aligned} \tag{6.23}$$

A port is responsive if it provides all resources given in the channel type, which is what the $\mathsf{prop}_{\mathbf{R}}$ elementary rule (Definition 4.8.2, page 46) states. For $r(tf)$, this is written $r_{\mathbf{R}} \lhd (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}})$, where the right hand side is just (6.5) from page 63 with $t$ and $f$ replacing 1 and 2. Composing with (6.23) reduces the dependency chain and we obtain $r_{\mathbf{R}} \lhd (b_{\mathbf{AR}} \wedge a_{\mathbf{AR}})$, as required.

## 6.5 Distributed Properties and $\tau$-Activeness

Most existential properties have elementary rules producing statements of the form $\bigwedge_i \gamma_i \lhd \top$, i.e. these resources are in some sense *local*, in that they are available at one particular point in the process. However this is not true of all resources. We already saw responsiveness as an example of universal resource that is provided through the collaboration of more than one part of a process, as in $\overline{a}\langle b\rangle \mid b.\mathbf{0}$ where $\overline{a}\langle b\rangle$ provides $a_{\mathbf{R}}$ conditional on $b_{\mathbf{A}}$ which is itself provided by $b.\mathbf{0}$.

The following proposition summarises what is required of distributed properties to ensure soundness. It is rather technical but basically enforces the semantics of $\lhd$: if an elementary rule produces a statement $\alpha \lhd \beta$ (with both $\alpha$ and $\beta$ existential) then composing with another process providing $\beta$ must yield a process in which $\alpha$ is immediately available without dependencies.

**Proposition 6.5.1 (Soundness of Distributed Properties)** *Let $\mathcal{K}$ be a set of properties including $\mathbf{R}$. Then $\Gamma \vdash_{\mathcal{K}} P$ implies $\Gamma \models P$ for all $\Gamma$ and $P$ if elementary rules for all $k \in \mathcal{K} \cap \mathcal{E}$ satisfy the following:*

*Let $p_k \lhd \bigwedge_{i\in I} \alpha_i \succeq \mathsf{prop}_k(G, \sigma, m, m')$ with $\alpha_i = p_{i k_i}$, and let $I_{\mathcal{E}} = \{i \in I : k_i \in \mathcal{E}\}$. Pick an arbitrary collection of guards $G_i$ (and types $\sigma_i$, multiplicities $m_i$, $m'_i$) with $i$ ranging $I_{\mathcal{E}}$ and $\mathsf{prop}_{k_i}(G_i, \sigma_i, m_i, m'_i) \succeq \alpha_i$. Then:*

$$\mathsf{good}_k(p \lhd \bigwedge_{i\in I\setminus I_{\mathcal{E}}} \alpha_i, (\Gamma \odot \bigodot_{i\in I_{\mathcal{E}}} \Gamma_i; G \mid \prod_{i\in I_{\mathcal{E}}} G_i))$$

This is proved as part of the Soundness proof, Section 7.6.

Although not that useful in practice, one example of a non-local existential resource is $\tau$-activeness, written $\tau_{\mathbf{A}}$, and meaning that the process will eventually do a $\tau$-transition.

For semantics, $\mathsf{good}_{\mathbf{A}}(\varepsilon \lhd \top, (\Gamma; P))$ holds if $P \xrightarrow{\tau} P'$ for some $P'$, in addition to what is given in Definition 6.3.1. The elementary rule sets

$$\mathsf{prop}_{\mathbf{A}}(G, \sigma, m, m') \overset{\text{def}}{=} \begin{cases} \mathsf{sub}(G)_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd \overline{\mathsf{sub}(G)}_{\mathbf{A}}) & \text{if } \#(G) = \omega \text{ or } m' \neq \star \\ \top & \text{otherwise} \end{cases}$$

One can easily verify this elementary rule satisfies the requirements of the above Proposition: The elementary rule only produces a non-local statement $\tau_{\mathbf{A}} \lhd \bar{p}_{\mathbf{A}}$ on a guard $G$ with subject $p$, and the only way to produce a $\bar{p}_{\mathbf{A}}$-resource is a guard $G'$ with subject $\bar{p}$. Composing the two processes as in the Proposition yields $G \mid G'$ which, using the (Com) rule of the labelled transition system, produces the $\tau$-transition $G \mid G' \rightarrow \mathbf{0}$, as required by the semantics.

For instance $\bar{a}$ has type $\bar{a}_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd a_{\mathbf{A}})$, and composing it with process $a$ produces the type $(\bar{a}_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd a_{\mathbf{A}})) \odot (a_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd \bar{a}_{\mathbf{A}})) \cong a_{\mathbf{A}} \wedge \bar{a}_{\mathbf{A}} \wedge \tau_{\mathbf{A}}$. If $a$ is linear, removal of unobservable dependencies returns just $\tau_{\mathbf{A}}$ for $a|\bar{a}$.

The typing rules make sure that the two complement guards eventually come to top-level so that they can communicate and produce a $\tau$-transition. One tricky counter-example is $a+\bar{a}$ which, using the (E-Sum) rule, produces

$$(a + \bar{a})_{\mathbf{A}} \wedge \big((\bar{a}_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd a_{\mathbf{A}})) \vee (a_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd \bar{a}_{\mathbf{A}}))\big)$$

where no further reduction can occur because for instance $\tau_{\mathbf{A}} \lhd a_{\mathbf{A}}$ and $a_{\mathbf{A}}$ are on different branches of the disjunction. Another tricky case is $a.\bar{a}$. The continuation has type $\bar{a}_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd a_{\mathbf{A}})$, which has the (E-PRE) gains the additional dependency $\mathsf{dep}_{\mathbf{A}}(a) = \bar{a}_{\mathbf{A}}$, becoming $\bar{a}_{\mathbf{A}} \lhd \bar{a}_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd (\bar{a}_{\mathbf{A}} \wedge a_{\mathbf{A}})) \cong \tau_{\mathbf{A}} \lhd (\bar{a}_{\mathbf{A}} \wedge a_{\mathbf{A}})$. Composing it with $\mathsf{prop}_{\mathbf{A}}(a, (), 1, 1) = a_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd \bar{a}_{\mathbf{A}})$ we get $a_{\mathbf{A}} \wedge (\tau_{\mathbf{A}} \lhd ((\bar{a}_{\mathbf{A}} \wedge a_{\mathbf{A}}) \vee \bar{a}_{\mathbf{A}})) \cong \tau_{\mathbf{A}} \lhd \bar{a}_{\mathbf{A}}$ which, by removal of the non-observable $\bar{a}_{\mathbf{A}}$-dependency, becomes $\tau_{\mathbf{A}} \lhd \bot$, or just $\top$.

# Chapter 7

# Structural Analysis

We will now leave universal and existential properties aside for a while and introduce a representation of process behaviour that is intermediary between the process syntax and a transition sequence, as summarised in Section 1.8 in the Introduction.

This framework is useful for proving soundness of the existential type system, provides a compact representation of liveness strategies, and is also useful as we'll see in the end of this section in deriving semantics and elementary rules for channel properties from a corresponding process-level property.

In order to keep track of the relation between behavioural statements and parts of process types we make two changes to processes, to enforce a certain structure making its analysis easier (without loss of generality, as every process is structurally congruent to a process of that form), and adding the dependency events mentioned in Section 5.4 into the process syntax. Specifically, an *annotated process* is any production from $P$ in the grammar below.

Extended names are used to distinguish between different private channels with the same name. for instance, using the standard $\pi$-calculus transition rules (ignoring the $\mathfrak{l}$-annotations), a $\tau$-transition on $a$ in $!\,a^{\mathfrak{l}}.(\boldsymbol{\nu} n)\,P \,|\, \bar{a}^{\mathfrak{l}'}$ would result in the process $!\,a^{\mathfrak{l}}.(\boldsymbol{\nu} n)\,P \,|\, (\boldsymbol{\nu} n)\,P$, that has two distinct channels with the same name $n$. Using extended names we can write the resulting process $!\,a^{\mathfrak{l}}.(\boldsymbol{\nu} n)\,P \,|\, (\boldsymbol{\nu} \mathfrak{l}'.\,n)\,P'$, where the extended name $\mathfrak{l}'.\,n$ gives information on how that binding was brought to top-level. An "extended event" similarly records what has happened to a given event annotation in the past.

Extended names and events are constructed by the labelled transition system on annotated processes (see page 85 and following). Up to that point the reader

$P \ ::= \ (\boldsymbol{\nu}\mathfrak{x})\,P \ \big| \ P_{\mathsf{par}}$

$P_{\mathsf{par}} \ ::= \ (P_{\mathsf{par}} \,|\, P_{\mathsf{par}}) \ \big| \ P_{\mathsf{sum}} \ \big| \ \mathbf{0}$

$P_{\mathsf{sum}} \ ::= \ (P_{\mathsf{sum}} + P_{\mathsf{sum}}) \ \big| \ G^{\mathfrak{l}}.P$

$G \ ::= \ !\,G_{\mathsf{norep}} \ \big| \ G_{\mathsf{norep}}$

$G_{\mathsf{norep}} \ ::= \ (\boldsymbol{\nu}\mathfrak{x})\,G_{\mathsf{norep}} \ \big| \ \bar{\mathfrak{a}}\langle\tilde{\mathfrak{x}}\rangle \ \big| \ \mathfrak{a}(\tilde{\mathfrak{y}})$

Extended event: $\mathfrak{l} \ ::= \ \mathfrak{l}.\,\mathfrak{l} \ \big| \ \bullet\,.\,\mathfrak{l} \ \big| \ l$

Extended name: $\mathfrak{a}, \mathfrak{x}, \mathfrak{y} \ ::= \ \mathfrak{l}.\,\mathfrak{x} \ \big| \ \bullet\,.\,\mathfrak{l} \ \big| \ x$

Table 7.1: Annotated Process Syntax

may assume that all $\mathfrak{x}$ and $\mathfrak{l}$ encountered are simple names and events "$x$" and "$l$".

In general we use the same letters $P$, $Q$, etc for both annotated processes and processes, and specify if a name corresponds to an annotated process in case of ambiguity.

**Definition 7.0.2 (Annotation Removal — Processes)** *Let $Q'$ be an annotated process.* Removing the event annotations *(written* $\mathrm{ran}(Q')$*) is done by repeatedly replacing every instance of $G^{\mathfrak{l}}.P$, and every extended name $\mathfrak{l}.x$ of the $P_{\mathsf{sum}}$ and $\mathfrak{x}$ rules in the grammar above by just $G.P$ (respectively, $x$).*

We will use the Barendregt convention on bound names as we will need to individually address bound channels by name:

**Definition 7.0.3 (Annotated Form)** *Let $Q$ be a process. An* annotated form *of $Q$ is any annotated process $Q'$ not using the same event more than once and such that all bound names are distinct from each other and from free names, such that* $\mathrm{ran}(Q') =_\alpha Q$.

**Example 7.0.4** *An annotated form of the process $P = (\boldsymbol{\nu} a)\,(a(x).\bar{x} \,|\, \overline{a}\langle b\rangle)$ is $P' = (\boldsymbol{\nu} a)\,(a(x)^{l_1}.\bar{x}^{l_2} \mid \overline{a}\langle b\rangle^{l_3})$, and* $\mathrm{ran}(P') = P$.

It is easy to see that every process has at least one annotated form, by $\alpha$-renaming bound names and for instance numbering all guards from left to right.

## 7.1 Strategies and Annotated Process Types

We modify the process types so that in some sense they contain the proof of their validity, by attaching strategies to every existential dependency statement.

Formally:

**Definition 7.1.1 (Liveness Strategy)** *Strategies are produced by the following grammar:*

$$\begin{array}{rcl}
\rho & ::= & \tilde{\pi}\lightning(\tilde{\pi})\delta \quad \big| \quad \pi\delta \quad \big| \quad \mathfrak{s} \\
\mathfrak{s} & ::= & \mathfrak{s}{+}\mathfrak{l} \quad \big| \quad \mathfrak{l} \\
\delta & ::= & .\,\rho \quad \big| \quad [\,s\,] \\
\pi & ::= & (\mathfrak{l}|\rho) \quad \big| \quad (\mathfrak{l}|\rho] \quad \big| \quad (\mathfrak{l}|\bullet) \quad \big| \quad (\rho|\bullet] \quad \big| \quad (\bullet|\rho) \\
\tilde{\pi} & ::= & \pi.\,\tilde{\pi} \quad \big| \quad \pi
\end{array}$$

*An* annotated existential dependency *is an expression of the form*

$$s_k \lhd \varepsilon \colon \rho,$$

*read "Strategy $\rho$ provides $s_k$ and depends on $\varepsilon$".*

Strategies refer to individual guards $G$ by the unique event $\mathfrak{l}$ they are attached to. And inversely statements such as "$\mathfrak{l}_1$ is at top-level" or "$\mathfrak{l}_1$ is $\mathfrak{l}_2$'s guard" refer to the attached guard, as in process $a(y)^{\mathfrak{l}_1}.\bar{b}\langle y\rangle^{\mathfrak{l}_2}$. A sum ${G_1}^{\mathfrak{l}_1}.Q_1 + {G_2}^{\mathfrak{l}_2}.Q_2$ is referred to by $\mathfrak{l}_1{+}\mathfrak{l}_2$. Formally:

**Definition 7.1.2 (Top-Level and Guards)** *A sum* $\mathfrak{s} = \sum_{i\in I} \mathfrak{l}_i$ *is* top-level *in a process* $P$ *if* $P \equiv (\boldsymbol{\nu}\tilde{z})\,(\sum_{j\in J} G_j{}^{\mathfrak{l}_j}.Q_j \mid R)$ *where* $\{\mathfrak{l}_i\}_{i\in I} = \{\mathfrak{l}_j\}_{j\in J}$.

*An event* $\mathfrak{l}$ guards *a sum* $\mathfrak{s}$ *in a process* $P$ *if* $P = C[G^{\mathfrak{l}}.Q]$ *where* $\mathfrak{s}$ *is at top-level in* $Q$.

A sequence $\pi_1 .\, \pi_2 .\, \cdots .\, \pi_n .\, \mathfrak{l}$ (abbreviated $\tilde{\pi}.\, \mathfrak{l}$) indicates how to bring a guard $\mathfrak{l}$ to top-level. An individual step $\pi_i = (\mathfrak{l}_i|\rho_i)$ tells to bring event $\mathfrak{l}_i$ to top-level, using $\rho_i$ to find a communication partner ($\rho_i = \bullet$ means the communication partner is to be found in the environment, i.e. $\mathfrak{l}_i$ should be brought to top-level with a *labelled* transition rather than a $\tau$). In that sequence, $\mathfrak{l}_1$ must be at top-level in the process, and $\mathfrak{l}_i$ must be $\mathfrak{l}_{i+1}$'s guard (This is enforced by *runnability*, cf. Definition 7.2.3). In such a sequence, a step $\mathfrak{l}$ can only appear at the end as it represents successful termination of a strategy, so $\mathfrak{l}.\, \rho$ is not a meaningful strategy.

The step $(\mathfrak{l}|\rho)$ in $(\mathfrak{l}|\rho).\, \rho'$ is said *doubly-anchored* (round bracket), meaning that both $\mathfrak{l}$ and $\rho$ must be accurately followed in order for that step to be successful. In contrast a *singly-anchored* step is written $(\mathfrak{l}|\rho].\, \rho'$ (square bracket) where the step is successful as soon as $\mathfrak{l}$ is consumed, even if not by communicating with $\rho$ (note that the left bracket is still round, to emphasise the fact that $\mathfrak{l}$ must be accurately followed, unlike $\rho$). Consider the process

$$P = a(y)^{l_a}.(\bar{s}^{l_{\bar{s}}} \mid \bar{y}^{l_{\bar{y}}}.\bar{t}^{l_{\bar{t}}}) \mid \overline{a}\langle b\rangle^{l_1} \mid \overline{a}\langle c\rangle^{l_2} \mid b^{l_b}\ . \tag{7.1}$$

In this example we named events according to their guard ports merely for readability — another convention would have to be used in case a port is used more than once in subject position.

One strategy for $\bar{s}$ is $(l_a|l_1].\, l_{\bar{s}}$ because it doesn't matter what $l_a$ is communicating with, as long as it is consumed. One strategy for $\bar{t}$ is $(l_a|l_1).\, (l_{\bar{y}}|l_b].\, l_{\bar{t}}$ because *a must* communicate with $\overline{a}\langle b\rangle$ labelled $l_1$ otherwise $\bar{y}$ won't get substituted to $\bar{b}$ and won't be able to communicate with $b$, preventing the next strategy step from occurring. On the other hand, if $\bar{y}$ communicates with some other $b$-input somewhere else, the strategy still works, so that second step is singly-anchored.

The expression $\pi_1 .\, \pi_2 .\, \cdots .\, \pi_n \lightning (\tilde{\pi}')\delta$ represents a strategy following the sequence of steps from $\pi_1$ to $\pi_n$ but, as it is about to consume step $\pi_n$, gets "hijacked" by a transition in a $P_j \xrightarrow{\tilde{\mu}_j} P_j'$ sequence from Definition 5.2.6. The $\tilde{\pi}'$ part is a sequence of steps forced by that sequence and is such that its last step prevents $\pi_n$ from taking place (for instance a step $(\mathfrak{l}|\rho_2)$ prevents a step $(\mathfrak{l}|\rho_1)$ if $\mathfrak{l}$ is not replicated). The $\delta$ tells how the strategy reacts to it.

Finally, $(\bullet|\mathfrak{l})\,[\,p\,]$, where $p$ is one of $n$ or $\bar{n}$ for some number $n$, tells to consume $\mathfrak{l}$ with a labelled transition, and that the required resource (whose liveness is being proved) is respectively the input or the output at $\mathfrak{l}$'s $n^{\text{th}}$ parameter. Note that such a step can't follow a step as in $(\mathfrak{l}_0|\rho).\, (\bullet|\mathfrak{l})\,[\,p\,]$, because that would mean that $\bullet$ is guarded by $\mathfrak{l}_0$, which is impossible as $\bullet$ is by definition in the environment and $\mathfrak{l}_0$ is in the process. Strategy $(\bullet|(\mathfrak{l}_0|\rho).\, \mathfrak{l})\,[\,p\,]$, on the other hand, is sensible ("use $\rho$ to consume $\mathfrak{l}_0$ and thereby bring $\mathfrak{l}$ to top-level, then consume $\mathfrak{l}$, to obtain liveness on its parameter port $p$").

**Example 7.1.3** *Consider the following process:*

$$P = \boldsymbol{!}\, t^{l_t} \mid \boldsymbol{!}\, a(x)^{l_a}.\bar{t}^{l_{\bar{t}}}.\bar{x}^{l_{\bar{x}}} \mid \overline{a}\langle b\rangle^{l_{\bar{a}}}.b^{l_b}.c^{l_c}.\bar{s}^{l_{\bar{s}}}$$

*which is an annotated form of* $\mathrm{ran}(P) = \mathbf{!}\,t \mid \mathbf{!}\,a(x).\bar{t}.\bar{x} \mid \overline{a}\langle b\rangle.b.c.\bar{s}$.

*The strategy for* $\bar{s}_{\mathbf{A}} \lhd \bar{c}_{\mathbf{A}}$ *is* $\rho = (l_{\bar{a}}|l_a).\,\big(l_b \,\big|\, (l_a|l_{\bar{a}}).\,(l_{\bar{t}}|l_t].\,l_{\bar{x}}\big].\,(l_c|\bullet].\,l_{\bar{s}}$ *(so the annotated dependency is* $\bar{s}_{\mathbf{A}} \lhd \bar{c}_{\mathbf{A}} : \rho$*). This strategy contains four steps, corresponding to the event stack* $l_{\bar{a}}$, $l_b$, $l_c$ *and* $l_{\bar{s}}$.

1. *The first step does a* $\tau_a$*-transition to bring* $l_{\bar{a}}$ *and* $l_a$ *to top-level.*
2. *In the second step,* $(l_a|l_{\bar{a}}).\,(l_{\bar{t}}|l_t].\,l_{\bar{x}}$ *tells how to find a communication partner for* $b$*, by first bringing the* $l_a$ *and* $l_{\bar{a}}$ *events to top-level (note that this step may seem redundant since it duplicates the previous step and doesn't correspond to an actual transition. However it may become necessary to unambiguously identify which instance of the replicated* $a$*-input we are talking about. So this* $l_{\bar{a}}$ *step really means we are going to work on the instance of* $\mathbf{!}\,a(x).\bar{t}.\bar{x}$ *that was created when* $l_{\bar{a}}$ *was brought to top-level, and not any other). The* $(l_{\bar{t}}|l_t]$ *step is a* $\tau$*-transition between* $\bar{t}$ *and* $t$*, and the final step* $l_{\bar{x}}$ *of the sub-strategy is our communication partner for* $b$ *consumed with a* $\tau$*-transition.*
3. *The third step* $(l_c|\bullet)$ *of the strategy indicates that* $c$*'s communication should be found in the environment, i.e. the strategy does a* $c$*-labelled transition at this point.*
4. *The final step* $l_{\bar{s}}$ *indicates where to find the* $\bar{s}$*, closing the liveness proof.*

*In this particular case, if* $a$ *is input plain, the dependency statement becomes* $\bar{s}_{\mathbf{A}} \lhd (\bar{c}_{\mathbf{A}} \wedge a_{\mathbf{R}})$*, which can be written* $(\bar{s}_{\mathbf{A}} \lhd \bar{c}_{\mathbf{A}}) \vee (\bar{s}_{\mathbf{A}} \lhd a_{\mathbf{R}})$*. The strategy for* $\bar{s}_{\mathbf{A}} \lhd a_{\mathbf{R}}$ *is*

$$(l_{\bar{a}}|l_a) \lightning (l_{\bar{a}}|\bullet).\,\big((l_b|(\bullet|l_{\bar{a}})[\,\bar{1}\,]].\,(l_c|\bullet].\,l_{\bar{s}}\big) :$$

*If a transition sequence* $\tilde{\mu}_i$ *from (5.2.6) consumes the* $a$*-output through the transition* $\xrightarrow{\overline{a}\langle b\rangle}$ *then it amounts to forcing* $\bar{a}$*'s communication partner to be* $\bullet$*, and the strategy on the right of the* $\lightning$ *is followed, doing a* labelled *transition* $\xrightarrow{b}$ *instead of* $\xrightarrow{\tau_b}$*. The* $b$*-output, communication partner of* $l_b$*, is obtained with* $(\bullet|l_{\bar{a}})[\,\bar{1}\,]$.

*Note that the strategy* $(l_{\bar{a}}|\bullet).\,\big((l_b|(\bullet|l_{\bar{a}})[\,\bar{1}\,]].\,(l_c|\bullet].\,l_{\bar{s}}\big)$ *on its own corresponds to the statement* $\bar{s}\lhd(\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle b\rangle)\wedge a_{\mathbf{R}})$ *— the strategy itself decided to do a labelled transition* $\xrightarrow{\overline{a}\langle b\rangle}$*, and therefore requires* $\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle b\rangle)$ *from the environment (in the* $\mathcal{K} = \mathbf{A}$*-case that's* $a_{\mathbf{A}}$*, activeness on* $a$*).*

*The following example shows more clearly how* $\mathsf{dep}_k$ *and responsiveness dependencies appear in strategies:*

$$P = \prod_{i\in I} {t_i}^{l_{ti}}.a(x)^{l_{ai}}.{u_i}^{l_{ui}}.\bar{x}^{l_{xi}} \mid \overline{a}\langle s\rangle^{l_{\bar{a}}}$$

*(where* $\mathrm{ran}(P) = \prod_{i\in I} t_i.a(x).u_i.\bar{x} \mid \overline{a}\langle s\rangle$*). That process satisfies the statement* $\bar{s}_{\mathbf{A}} \lhd \bigvee_{i\in I} \bigwedge_{j\in I} (\bar{t}_{i\mathbf{A}} \wedge \bar{u}_{j\mathbf{A}})$ *or, equivalently,*

$$\bigvee_{j\in I} \bigwedge_{i\in I} \big(\bar{s}_{\mathbf{A}} \lhd (\bar{t}_{i\mathbf{A}} \wedge \bar{u}_{j\mathbf{A}})\big) \tag{7.2}$$

*Any strategy for* $\bar{s}_{\mathbf{A}}$ *can* choose *an* $i \in I$ *(the communication partner it will select for* $a$ *in the absence of interference), which causes the dependency on* $\bar{t}_{i\mathbf{A}}$*, but,* *in an actual run, it can be* forced *a connection with the a-input corresponding to any $j \in I$, after which it will require $\bar{u}_{j\mathbf{A}}$. The strategy for that scenario is $\rho_{ij} = (l_{ti}|\bullet].\,(l_{ai}|l_{\bar{a}})\lightning\big((l_{tj}|\bullet].\,(l_{aj}|l_{\bar{a}})\big).\,(l_{uj}|\bullet].\,l_{xj}$ and depends on $\bar{t}_{i\mathbf{A}} \wedge \bar{u}_{j\mathbf{A}}$: The strategy prepares a communication between $l_{\bar{a}}$ and $l_{ai}$ by consuming the $t_i$-prefix (which causes the dependency on $\bar{t}_{i\mathbf{A}}$). But then the communication on $\bar{a}$ is hijacked with the sequence $\xrightarrow{t_j}\xrightarrow{\tau}$ where the latter transition consumes $l_{aj}$. Note how the two corresponding steps are grouped by brackets in the strategy to distinguish the part caused by external interference (not creating dependencies) and the strategy's reaction, which is to consume the $u_j$-prefix (causing a $\bar{u}_{j\mathbf{A}}$-dependency) to bring $\bar{s}$ to top-level.*

*Inserting strategies $\rho_{ij}$ into the behavioural statement (7.2) gives the following* annotated statement *for P:*

$$\bigvee_{j\in I}\bigwedge_{i\in I}\big(\bar{s}_{\mathbf{A}} \lhd \big(\bar{t}_{i\mathbf{A}} \wedge \bar{u}_{j\mathbf{A}}\big)\colon \big((l_{ti}|\bullet].\,(l_{ai}|l_{\bar{a}})\lightning\big((l_{tj}|\bullet].\,(l_{aj}|l_{\bar{a}})\big).\,(l_{uj}|\bullet].\,l_{xj}\big)\big)$$

We will often set conditions on strategies *and strategies they contain.* The following definition makes that concept precise.

**Definition 7.1.4 (Sub-strategies)** *The* contains *relation is the least transitive relation on liveness strategies such that:*

- *Let $\rho = \pi_1.\,\cdots.\,\pi_{n-1}.\,\mathfrak{s}$ where $\pi_i \in \{(\mathfrak{l}_i|\rho_i), (\mathfrak{l}_i|\rho_i]\}$. Then, for all $1 \le i < n$, $\rho$ contains $\rho_i$ and $\pi_1.\,\cdots.\,\mathfrak{l}_i$.*
- *Let $\rho = \pi_1.\,\cdots.\,(\mathfrak{l}_n|\rho_n)\lightning(\tilde{\pi}')\delta$. Then $\rho$ contains $(\pi_1.\,\cdots.\,\mathfrak{l}_n)$, $\rho_n$ and $\tilde{\pi}'\delta$.*
- *Let $\rho = (\bullet|\rho_0)\,[\,s\,]$. Then $\rho$ contains $\rho_0$.*

*If $\rho$ contains $\rho_0$, the latter is called a* sub-strategy *of the former.*

Just like liveness strategies prove correctness of an existential dependency statement, responsiveness strategies prove correctness of a responsiveness statement. The idea is to attach a strategy to every element of the behavioural statement as found in the channel type:

**Definition 7.1.5 (Responsiveness Strategy)** *A* responsiveness strategy *is an expression $\rho.\,\phi$ where $\phi$ is generated by the following grammar:*

$$\phi \;::=\; s_k : \rho \;\;\big|\;\; p_{\mathbf{R}} : \rho.\,\phi \;\;\big|\;\; p_{\mathbf{R}} : \bullet \;\;\big|\;\; \phi \vee \phi \;\;\big|\;\; \phi \wedge \phi \;\;\big|\;\; \top \;\;\big|\;\; \bullet$$

*where k ranges over $\mathcal{E}$, p over numerical ports and s over sums of numerical ports.*

**Definition 7.1.6 (Annotated Responsiveness Statement)** Removing the annotations *of a responsiveness strategy $\phi$ (in which $\bullet$ may only appear behind a "$\gamma$:" prefix), written* $\mathrm{ran}(\phi)$, *is the logical homomorphism yielding a dependency $\varepsilon$ such that*

- $\mathrm{ran}(s_k : \rho) \overset{\mathrm{def}}{=} s_k$
- $\mathrm{ran}(p_{\mathbf{R}} : \rho.\,\phi) \overset{\mathrm{def}}{=} p_{\mathbf{R}}$

*Let $p$ be a port and $\xi$ the corresponding behavioural statement in the channel type. Then an* annotated responsiveness statement *for $p$ is an expression of the form $p_{\mathbf{R}} \lhd \varepsilon\colon \rho.\,\phi$ where* $\mathrm{ran}(\phi) = \xi$*.*

In $p_{\mathbf{R}} \lhd \varepsilon\colon \rho.\,\phi$, $\rho$ tells precisely which $p$-prefix is being talked about, and $\phi$ gives the strategies of its parameters, if any.

Note the distinction, in a responsiveness strategy, between $p_{\mathbf{R}} : l.\top$ and $p_{\mathbf{R}} : \bullet$ — the former occurs for channels with trivial behavioural statements (e.g. parameterless channels), therefore always responsive, and the latter occurs for channels that do not appear in a process, and are therefore vacuously responsive.

Delegated responsiveness such as $b$ in $\overline{a}\langle b\rangle^{l}$ is expressed with statements like $b_{\mathbf{R}} \lhd a_{\mathbf{AR}} : (\bullet|l)\,[1]\,.\,\bullet$ where $(\bullet|l)\,[1]$ specifies any remote use of $b$ and $\bullet$ indicates that responsiveness is provided by the environment. Compare with $b_{\mathbf{A}} \lhd a_{\mathbf{AR}}\colon (\bullet|l)\,[1]$ that represents remote *activeness* on $b$.

Liveness and responsiveness strategies can be put together as follows:

**Definition 7.1.7 (Annotated Behavioural Statement)** *An annotated behavioural statement (ranged over by $\Phi$) follows the grammar for $\Delta$ given in (1.2) on page 6, but where the $\gamma \lhd \varepsilon$ rule is replaced by annotated statements ($k \in \mathcal{E}$)*

$$\cdots \quad \Big| \quad s_k \lhd \varepsilon\colon \rho \quad \Big| \quad p_{\mathbf{R}} \lhd \varepsilon\colon \rho.\,\phi \quad \Big| \quad \cdots$$

**Definition 7.1.8 (Annotated Process Type)** *An annotated process type is a structure of the form $(\Sigma; \Phi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ where $\Phi_{\mathrm{L}}$ is an annotated behavioural statement, and $\Xi_{\mathrm{E}}$ a behavioural statement.*

*Removing strategy annotations from an annotated process type is again done by the* ran *operator, that recursively replaces $s_k \lhd \varepsilon\colon \rho$ and $p_{\mathbf{R}} \lhd \varepsilon\colon \rho.\,\phi$ by $s_k \lhd \varepsilon$ and $p_{\mathbf{R}} \lhd \varepsilon$, respectively.*

## 7.2 Structural Semantics: Consistency

In this section we provide precise semantics for annotated behavioural statements. Semantics are split into two parts. Consistency requires a strategy to only attempt making two prefixes communicate if they have complement ports and are at top-level. Completeness in Section 7.3 requires to have a strategy for every possible interference.

The $\mathsf{sub}$ operator gives the port brought to top-level by the given strategy, the $\mathsf{obj}$ operator gives the objects of the prefix brought to top-level, and $\mathsf{subst}_P(\pi)$ is the name substitution applied by a strategy step $\pi$.

For instance, having $P = a(x)^{l_a}.\bar{b}\langle x\rangle^{l_b} \,|\, \overline{a}\langle t\rangle^{l_{\bar{a}}}$:

$\mathsf{sub}_P((l_a|l_{\bar{a}}).\,l_{\bar{b}}) = \bar{b}$, $\mathsf{obj}_P((l_a|l_{\bar{a}}).\,(l_{\bar{b}}|\bullet)) = t$, and $\mathsf{subst}_P((l_a|l_{\bar{a}})) = \{{}^{t}\!/\!{}_{x}\}$.

Special care is required for bound names, as a single (bound) name can refer to more than one actual channel over the run of a process. For instance, having $P = \,!\,a^{l_a}.(\boldsymbol{\nu} n)\,Q \,|\, \bar{a}^{l_1}.Q_1 \,|\, \bar{a}^{l_2}.Q_2$, there are two distinct channels $n$ to be considered, for each $i$, the one brought to top-level with $Q_i$, which is the value of $\mathsf{sub}((l_a|l_i).\,\rho)$ (assuming $\mathsf{sub}(\rho) = n$). These two $n$-channels are written $(l_a|l_i).\,\boldsymbol{\nu} n$ for $i \in \{1, 2\}$.

More generally, separate instances of a bound name $x$ are identified by prefixing $\boldsymbol{\nu} x$ with a sequence of strategy steps $\tilde{\pi}$.

Such "extended port names" obey the following grammar:

$$\mathfrak{p} \ ::= \ \pi.\,\mathfrak{p} \ \big| \ \boldsymbol{\nu} p \ \big| \ p$$

Quotiented by the congruence given by the relation $\forall \pi, p : \pi.\,p \mapsto p$ (i.e. only bound names may be prefixed).

The complement $\bar{\mathfrak{p}}$ of such an extended port is obtained with $\overline{\pi.\,\mathfrak{p}} \overset{\text{def}}{=} \pi.\,\bar{\mathfrak{p}}$ and $\overline{\boldsymbol{\nu} p} \overset{\text{def}}{=} \boldsymbol{\nu}\bar{p}$.

**Definition 7.2.1 (Strategy Subject and Objects)** *Let $P$ be a process of the form $C[(\boldsymbol{\nu}\tilde{z})\,(R \mid G^{\mathfrak{l}}.Q)]$.*

*The* subject *of a strategy $\rho$ in $P$ is a port written $\mathsf{sub}_P(\rho)$. The* objects *of a sequence of steps $\tilde{\pi}$ in $P$ is a name sequence written $\mathsf{obj}_P(\tilde{\pi})$.*

*The* substitution *associated with a sequence of steps $\tilde{\pi}$ in $P$ is a function mapping names to extended ports written $\mathsf{subst}_P(\tilde{\pi})$. We write $\mathsf{subst}_P(\tilde{\pi})a$ to apply the function $\mathsf{subst}_P(\tilde{\pi})$ on name $a$. By extension, $\mathsf{subst}_P(\tilde{\pi})\mathfrak{p}$ is the identity if $\mathfrak{p}$ is not a free port, and $\mathsf{subst}_P(\tilde{\pi})a$ if $\mathfrak{p} = \bar{a}$. It acts on each extended port individually when passed a tuple as in $\mathsf{subst}_P(\tilde{\pi})\tilde{\mathfrak{p}}$. Finally, a substitution applies individually to each free name when given entire guards as in $\mathsf{subst}_P(\tilde{\pi})G$.*

*These three functions are defined inductively on $\rho$, according to the following rules:*

1. $\mathsf{sub}_P(\mathfrak{l}) \overset{\text{def}}{=} (\boldsymbol{\nu}\tilde{z})\,\mathsf{sub}(G)$ *(where $(\boldsymbol{\nu}\tilde{z})\,\mathfrak{p}$ is $\boldsymbol{\nu} p$ if $\mathfrak{p} = p$ and $\mathrm{n}(p) \in \tilde{z}$, $\mathfrak{p}$ otherwise).*

2. $\mathsf{sub}_P(\tilde{\pi}.\,\mathfrak{l}) = \mathsf{subst}_P(\tilde{\pi})\mathsf{sub}_P(\mathfrak{l})$

3. $\mathsf{sub}_P(\pi\,[\,p\,]) \overset{\text{def}}{=} \mathsf{obj}_P(\pi)\,[\,p\,]$
   *(where $(\mathfrak{x}_1, \ldots, \mathfrak{x}_n)\,[\,i\,] = \mathfrak{x}_i$ and $(\mathfrak{x}_1, \ldots, \mathfrak{x}_n)[\,\bar{\imath}\,] = \overline{\mathfrak{x}_i}$)*

4. $\mathsf{sub}_P(\tilde{\pi}\lightning(\tilde{\pi}')\delta) \overset{\text{def}}{=} \mathsf{sub}_P(\tilde{\pi}'\delta)$

5. $\mathsf{obj}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho)) \overset{\text{def}}{=} \mathsf{subst}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho))\mathsf{obj}(G)$.

6. $\mathsf{subst}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho))a \overset{\text{def}}{=} a$ *if* $a \notin (\mathrm{bn}(G) \cup \tilde{z})$

7. $\mathsf{subst}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho))a \overset{\text{def}}{=} \tilde{\pi}.\,\boldsymbol{\nu} a$ *if* $a \in \tilde{z}$

8. $\mathsf{subst}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho))(\mathsf{obj}(G)\,[\,i\,]) \overset{\text{def}}{=} \mathsf{obj}_P((\rho|\tilde{\pi}.\,\mathfrak{l}))\,[\,i\,]$ *if $G$ is an input and if* $\rho \neq \bullet$.

9. $\mathsf{subst}_P(\tilde{\pi}.\,\pi)a \overset{\text{def}}{=} \tilde{\pi}.\,\pi.\,\boldsymbol{\nu} a$ *if $a \in \mathrm{bn}(G)$ and either $\pi = (\mathfrak{l}|\rho]$ for some $\rho$, or $\pi = (\mathfrak{l}|\bullet)$, or $G$ is an output and $\pi = (\mathfrak{l}|\rho)$ for some $\rho$.*

10. *All operators used in this definition commute with sums, e.g.*

$$\mathsf{sub}_P(\sum_i \mathfrak{l}_i) \overset{\text{def}}{=} \sum_i \mathsf{sub}_P(\mathfrak{l}_i)$$

$$(\boldsymbol{\nu}\tilde{z})\,\sum_i \mathfrak{p}_i \overset{\text{def}}{=} \sum_i (\boldsymbol{\nu}\tilde{z})\,\mathfrak{p}_i$$

$$\mathsf{sub}_P(\pi\left[\sum_i \mathfrak{l}_i\right]) \overset{\text{def}}{=} \sum_i \mathsf{sub}_P(\pi\,[\,\mathfrak{l}_i\,])$$

We omit the index $P$ when there is no ambiguity.

As said earlier, one application of strategies is to prove availability of an existential resource $p_k$, i.e. that following the strategy brings to top-level a guard $G^{\mathfrak{l}}$ that $\mathsf{prop}_k$ declares to provide resource $p_k$. Assume, without loss of generality (see below) that elementary rules all produce a single statement of the form $p_k \triangleleft \varepsilon$, where $p$ and $\varepsilon$ are computed depending on the guard or sum it is applied to.

As $\mathsf{prop}_k$ commutes with substitution (Definition 4.4.1), $\mathrm{n}(p)$ either belongs to $\mathsf{sub}(G) \cup \mathsf{obj}(G)$ or is a "fake" port such as $\mathsf{proc}$ or $\tau$. We can define accordingly a *target* function:

**Definition 7.2.2 (Target Function)** *Let $k$ be a property whose elementary guard (respectively, sum) rule is of the form $\mathsf{prop}_k(G, \sigma, m, m') = p_k \triangleleft \varepsilon$, where $p$ depends on $G$ and $\varepsilon$ on any of $G$, $\sigma$, $m$ and $m'$. Then the corresponding* target function $\mathsf{trg}_{k,P}$*:*

- *maps event labels to resources such that $\mathsf{prop}_k(G, \sigma, m, m') = \mathsf{trg}_{k,P}(\mathfrak{l})_k \triangleleft \varepsilon$ for some $\varepsilon$ if $G^{\mathfrak{l}}$ appears in $P$.*
- *The target of a sum $\mathsf{trg}_{k,P}(\mathfrak{l}_1 + \mathfrak{l}_2 + \dots)$ similarly extracts the resource from the elementary sum rule.*
- *A target function is generalised to arbitrary strategies (writing $\mathsf{trg}_{k,P}(\rho)$) by applying substitutions as in Definition 7.2.1.*
- *The target of a delegating strategy is given by*

$$\mathsf{trg}_{k,P}((\bullet|\rho)\,[\,s\,]) \overset{\text{def}}{=} \mathsf{sub}_P((\bullet|\rho)\,[\,s\,])$$

Elementary rules commuting with substitutions implies

$$\mathsf{trg}_{k,G\{\tilde{x}/\tilde{y}\}}(\mathfrak{l}) = \mathsf{trg}_{k,G}(\mathfrak{l})\{\tilde{x}/\tilde{y}\}$$

The case of elementary properties producing more than one statement is obtained by splitting the properties (for instance the $\tau$-activeness elementary guard rule produces statement of the form $\tau_{\mathbf{A}} \triangleleft \varepsilon \wedge p_{\mathbf{A}} \triangleleft \varepsilon'$, but it can be split into two existential properties $\mathbf{A}$ and $\mathbf{A}'$, having respectively elementary guard rules of the form $\tau_{\mathbf{A}} \triangleleft \varepsilon$ and $p_{\mathbf{A}'} \triangleleft \varepsilon'$. Forking $\mathbf{A}$ into two resources $\mathbf{A}$ and $\mathbf{A}'$ is only required to know which part of the elementary rule the strategy is interested in, and the same result could be obtained by including this information in liveness strategy themselves, as in "$\rho$ provides resource $\gamma$ by using the $i^{\text{th}}$ factor of $k$'s elementary rule". For properties studied in this thesis, $\mathsf{trg}_{k,P}(\rho)$ is one of $\mathsf{sub}_P(\rho)_k$, $\tau_k$, $\mathsf{proc}_k$ (Section 7.7) and $s_k$ where $s$ is a sum (Chapter 6).

As a first step to deciding correctness of a strategy, the following definition tells whether a strategy for an liveness resource $\gamma$ is actually able to bring its target guard to top-level in the absence of interference. Note that it is not really useful as is because a strategy may in some way interfere with itself (e.g. in $(l_1|\rho_1).\,(l_2|\rho_2).\,l_3$, $\rho_1$ could interfere with $\rho_2$). On the other hand, this notion combined with the *completeness* introduced in the next section becomes sufficient for correctness of a type.

In the fourth point, let $\mathsf{sub}_P(\rho) = p$, $\mathrm{n}(p) = a$ and $\Sigma(a) = (\tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}})$ ($\Sigma$ being $\Gamma$'s channel type mapping). Then $\Sigma_P(\rho)$ is $\xi_{\mathrm{I}}$ if $p = a$ and $\xi_{\mathrm{O}}$ if $p = \bar{a}$.

**Definition 7.2.3 (Runnable Strategy)** *Let $(\Gamma; P)$ be a typed process. Then a strategy is $(\Gamma; P)$-*runnable *if and only if it satisfies all the following rules:*

- *A strategy is only $(\Gamma; P)$-runnable if all its sub-strategies are also $(\Gamma; P)$-runnable.*
- *A strategy $\mathfrak{s}$ is $(\Gamma; P)$-runnable if $\mathfrak{s}$ is at top-level in $P$ in the sense of Definition 7.1.2.*
- *For a strategy $\tilde{\pi}.\,(\mathfrak{l}|\rho).\,\mathfrak{s}$, let $\mathfrak{p} = \mathsf{sub}_P(\tilde{\pi}.\,\mathfrak{l})$. Then:*
  - *$\mathfrak{l}$ guards $\mathfrak{s}$ in $P$, in the sense of Definition 7.1.2.*
  - *If $\rho = \bullet$: $\mathfrak{p} = p$ for some $p$ and $\Gamma{\downarrow_p}$*
  - *If $\rho \neq \bullet$: $\mathsf{sub}_P(\rho) = \bar{\mathfrak{p}}$.*
- *For a strategy $(\bullet|\rho)\,[\,s\,]$, $\mathsf{sub}_P(\rho) = p$ for some $p$, $\Gamma{\downarrow_p}$ and $\Sigma_P(\rho){\downarrow_s}$*
- *For a strategy $\tilde{\pi}.\,(\mathfrak{l}|\rho)\lightning(\tilde{\pi}'.\,(\mathfrak{l}'|\rho'))\delta$, $\tilde{\pi}.\,(\mathfrak{l}|\rho)$ is runnable (checked by ignoring the condition on $\mathfrak{s}$ in first point), and either $\rho = \rho'$ or $\tilde{\pi}.\,\mathfrak{l} = \tilde{\pi}'.\,\mathfrak{l}'$.*

Note how the semantics of "$(\mathfrak{l}|\rho)$" versus "$(\mathfrak{l}|\rho]$" affect runnability through the definition of $\mathsf{sub}$. The substitution $\mathsf{subst}(\pi)$ is only applied to subsequent objects and subjects when $\pi$ is doubly anchored (cf. Definition 7.2.1). Therefore, in process (7.1) on page 75, strategy $(l_a|l_1].\,(l_{\bar{y}}|l_b].\,l_{\bar{t}}$ is *not* runnable because the first step is singly-anchored and so doesn't apply a substitution on its object $y$, and so, in the next step, (extended) ports $(l_a|l_1).\,\boldsymbol{\nu}\bar{y}$ and $b$ aren't complements. On the other hand, strategy $(l_a|l_1).\,(l_{\bar{y}}|l_b].\,l_{\bar{t}}$ is runnable because now the first step applies the substitution $\{{}^{b}\!/\!{}_{y}\}$, so ports in the next step become complements ($\bar{b}$ and $b$), as required.

If a strategy $\rho$ with target $\gamma$ is runnable then there is some $\varepsilon$ such that $\gamma \lhd \varepsilon : \rho$ is correct in absence of interference. The following definition gives a lower bound (proved as a part of Lemma 7.4.10) on $\varepsilon$. It builds on the $\mathsf{dep}_{\mathcal{K}}$ operator (Definition 5.2.5). Dependency $\mathsf{dep}^{-}_{\mathcal{K},P}(\rho)$ gives the resources required to bring $\rho$'s target to top-level, while $\mathsf{dep}_{\mathcal{K},P}(\rho)$ gives the resources required to consume its target with a labelled transition.

**Definition 7.2.4 (Dependencies of a Strategy)** *Let $P$ be a process and $\rho$ a runnable strategy.*

*Then $\rho$'s $\mathcal{K}$-*dependencies *with respect to $P$, written $\mathsf{dep}^{-}_{\mathcal{K},P}(\rho)$, is the dependency $\varepsilon$ defined as follows.*

- $\mathsf{dep}_{\mathcal{K},P}(\mathfrak{l}) \overset{\text{def}}{=} \mathsf{dep}_{\mathcal{K}}(G)$ *if $P = C[G^{\mathfrak{l}}.Q]$ for some $Q$ and $C[\cdot]$.*
- *The general case of $\mathsf{dep}_{\mathcal{K},P}(\rho)$ builds on the previous rule like Definition 7.2.1 does for $\mathsf{sub}_P$.*
- $\mathsf{dep}^{-}_{\mathcal{K},P}(\mathfrak{s}) \overset{\text{def}}{=} \top$
- $\mathsf{dep}^{-}_{\mathcal{K},P}((\bullet|\rho)\,[\,s\,]) \overset{\text{def}}{=} \mathsf{dep}^{-}_{\mathcal{K},P}(\rho) \wedge \mathsf{dep}_{\mathcal{K},P}(\rho) \wedge \overline{\mathsf{sub}_P(\rho)}_{\mathbf{R}}$
- $\mathsf{dep}^{-}_{\mathcal{K},P}((\mathfrak{l}|\bullet)) \overset{\text{def}}{=} \mathsf{dep}^{-}_{\mathcal{K},P}((\mathfrak{l}|\bullet]) \overset{\text{def}}{=} \mathsf{dep}_{\mathcal{K},P}(\mathfrak{l})$
- $\mathsf{dep}^{-}_{\mathcal{K},P}((\mathfrak{l}|\rho)) \overset{\text{def}}{=} \mathsf{dep}^{-}_{\mathcal{K},P}((\mathfrak{l}|\rho]) \overset{\text{def}}{=} \mathsf{dep}^{-}_{\mathcal{K},P}(\rho)$

- $\mathsf{dep}^{-}_{\mathcal{K},P}(\pi.\,\rho) \overset{\text{def}}{=} \mathsf{dep}^{-}_{\mathcal{K},P}(\pi) \wedge \mathsf{dep}^{-}_{\mathcal{K},P}(\rho).$
- $\mathsf{dep}^{-}_{\mathcal{K},P}(\tilde{\pi}.\,(\mathfrak{l}|\rho)\lightning(\tilde{\pi}').\,\rho_2) \overset{\text{def}}{=} \mathsf{dep}^{-}_{\mathcal{K},P}(\tilde{\pi}.\,\mathfrak{l}) \wedge \mathsf{dep}^{-}_{\mathcal{K},P}(\rho_2)$
- $\mathsf{dep}^{-}_{\mathcal{K},P}(\tilde{\pi}.\,(\mathfrak{l}|\rho)\lightning(\bullet|\rho')\,[\tilde{p}]) \overset{\text{def}}{=} \mathsf{dep}^{-}_{\mathcal{K},P}(\tilde{\pi}.\,\mathfrak{l}) \wedge \overline{\mathsf{sub}_P(\rho')}_{\mathbf{R}}$

In the next to last point, $\tilde{\pi}'$ is irrelevant when computing a strategy's dependencies. The reason is that $\mathsf{dep}$ computes what is required by the strategy to progress on its own, while $\tilde{\pi}'$ represent interference being forced upon it. In the last point the strategy requires remote *responsiveness* but not $\mathsf{dep}_{\mathcal{K},P}(\rho')$, for the same reason.

For responsiveness, the dependencies are obtained by putting together the dependencies of every component in the strategy. Note how it requires the responsiveness strategy $\phi$ to closely follow the structure of the behavioural statement $\xi$.

**Definition 7.2.5 (Dependencies of a Responsiveness Strategy)** *Let $p$ be a port whose parameter types are $\tilde{\sigma}$, and whose behavioural statement in the channel type is $\xi$. We define $\tilde{\sigma}_i$ and $\xi_q$ so that $\sigma_i = (\tilde{\sigma}_i; \xi_i; \xi_{\bar{\imath}})$.*

*The dependencies $\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\xi,\phi)$ of a responsiveness strategy $\phi$ for $p$ is inductively obtained as follows:*

- $\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\top,\top) = \top$
- $\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\xi_1 \vee \xi_2,\phi_1 \vee \phi_2) = \mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\xi_1,\phi_1) \vee \mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\xi_2,\phi_2)$
- $\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\xi_1 \wedge \xi_2,\phi_1 \wedge \phi_2) = \mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\xi_1,\phi_1) \wedge \mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\xi_2,\phi_2)$
- $\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},s_k \lhd \varepsilon, s_k\colon \rho) = \mathsf{dep}^{-}_{\mathcal{K},P}(\rho) \setminus \varepsilon$
- $\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},q_{\mathbf{R}} \lhd \varepsilon, q_{\mathbf{R}}\colon \rho.\,\phi) = \mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma}_{\mathrm{n}(q)},\xi_q,\phi) \setminus \varepsilon$
- $\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma},\gamma \lhd \varepsilon, \bullet) = \gamma$

Runnability is lifted to process types, by requiring each of its strategies to be runnable and respect the declared dependencies:

**Definition 7.2.6 (Consistent Typed Process)** *An annotated typed process $(\Gamma; P)$ is said* consistent *with respect to a set of existential properties $\mathcal{K}$ if*

- *for every liveness statement $s_k \lhd \varepsilon\colon \rho$ in $\Gamma$'s local component, $\rho$ is runnable for $P$, $\mathsf{dep}^{-}_{\mathcal{K},P}(\rho) \succeq \varepsilon$ and $\mathsf{trg}_{k,P}(\rho) = s_k$.*
- *for every responsiveness statement $p_{\mathbf{R}} \lhd \varepsilon\colon \rho.\,\phi$ in $\Gamma$'s local component, for every liveness strategy $\rho'$ appearing in $\phi$, $(\rho|\bullet).\,\rho'$ is runnable for $P$ and $\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma}_{\mathrm{n}(p)},\xi_p,\phi) \succeq \varepsilon$, writing $(\tilde{\sigma}_a; \xi_a; \xi_{\bar{a}})$ for the type of a channel $a$ in $\Gamma$.*

## 7.3 Structural Semantics: Completeness

There are two forms of choices that a process (whether it is selection or branching) can do. The most obvious is the $\pi$-calculus sum operator $P{+}Q$ that can evolve according to $P$ or according to $Q$. The second form is obtained by having a non-replicated prefix having more than one possible communication partner, as in

$$(\boldsymbol{\nu} a)\,(a(x)^{l_a}.\bar{x}.\bar{s} \mid \overline{a}\langle b\rangle^{l_b} \mid \overline{a}\langle c\rangle^{l_c} \mid P) \tag{7.3}$$

(where $P$ provides $b$ and $c$ in some way). In that process, there should be (at least) two activeness strategies for $\bar{s}_{\mathbf{A}}$, one in case $a$ connects to $\overline{a}\langle b\rangle$ and one in case it is $\overline{a}\langle c\rangle$.

For both forms, every possible choice should be taken into account in separate components of the behavioural statement, these components being separated by $\vee$-connectives. Consider for example an liveness strategy $\tilde{\pi}.\,(\mathfrak{l}|\rho)\delta$ where $\mathfrak{l}$ could find partners $\rho_i$ other than $\rho$. There should then be as many $\tilde{\pi}.\,(\mathfrak{l}|\rho)\lightning(\mathfrak{l}|\rho_i)\delta_i$, again separated by $\vee$-connectives. Note that $\mathfrak{l}$'s communication partner is effectively a selection performed by the process.

We now give a way to accurately describe choices made by a process or its environment over a particular run. Consider a process $P = \sum_i {G_i}^{l_i}.P_i$. The type of that process is essentially $\bigvee_i \Gamma_i$ where each $\Gamma_i$ corresponds to one term of the sum. We identify one particular choice with the corresponding event $l_i$:

**Definition 7.3.1 (Sum Guard)** *An event $\mathfrak{l}$ is a* sum guard *in a process $P$ if $P = C[\sum_{i\in I} {G_i}^{\mathfrak{l}_i}.Q_i]$ and $\mathfrak{l} = \mathfrak{l}_i$ for some $i \in I$.*

*Two distinct events $\mathfrak{l}_1$ and $\mathfrak{l}_2$ are* contradicting *sum guards if they satisfy the above for the same context $C[\cdot]$, event and process sets $Q_i$, $\mathfrak{l}_i$, but different $i \in I$.*

If the sum itself is guarded, we identify a choice with a *strategy* $\rho$ (called a *selection strategy*). For instance in

$$Q = \boldsymbol{!}\,a(\tilde{y})^{l_a}.P \,|\, \overline{a}\langle\tilde{x}\rangle^{l} \,|\, \overline{a}\langle\tilde{z}\rangle^{l'} \tag{7.4}$$

where $P$ is as in Definition 7.3.1, independent choices will be made for each $a$-output, and are identified by expressions of the form $(l_a|l).\,l_i$ or $(l_a|l').\,l_i$, respectively. Selections made by third-party processes are identified in a similar way. For instance in a process $\overline{a}\langle tf\rangle^{l}$, $a$ being a Boolean channel (see Introduction), it is assumed (and described in the channel type) that the environment will select one of $\bar{t}_{\mathbf{A}}$ and $\bar{f}_{\mathbf{A}}$. As the reader will expect, those two choices are respectively described as $(\bullet|l)[\,\overline{1}\,]$ and $(\bullet|l)[\,\overline{2}\,]$.

Choice of a communication partner is written as a pair $(\mathfrak{l}|\rho)$. For instance (7.3) has two selection strategies $(l_a|l_b)$ and $(l_a|l_c)$. In case $\mathfrak{l}$ is not at top-level selection strategies take the form $\pi_1.\,\cdots.\,\pi_n$.

The complete set of choices made by a process over a particular course can be described by a set of such selection strategies. For instance (7.4) has four possible choice sets, all of the form $\{\,(l_a|l).\,l_i,\;(l_a|l').\,l_j\,\}$ where $i$ and $j$ independently range over 1 and 2.

The following two definitions clarify some concepts needed to precisely define contradicting strategies.

**Definition 7.3.2 (Matching Steps)** *Two sequences of steps $\pi_1. \cdots .\pi_n$ and $\pi'_1. \cdots .\pi'_n$ (where $\pi_i \in \{(\mathfrak{l}_i|\rho_i), (\mathfrak{l}_i|\rho_i]\}$ and $\pi'_i \in \{(\mathfrak{l}'_i|\rho'_i), (\mathfrak{l}'_i|\rho'_i]\}$)* match *if for all $1 \leq i \leq n$: $\mathfrak{l}_i = \mathfrak{l}'_i$ and either $\rho_i = \rho'_i$ or (at least) one of $\pi_i$ and $\pi'_i$ is singly-anchored.*

*Two sequences $\tilde{\pi}_1$ and $\tilde{\pi}_2$ are* equivalent *if any sequence $\tilde{\pi}$ matches $\tilde{\pi}_1$ if and only if it matches $\tilde{\pi}_2$.*

Sequences are equivalent if and only if they only differ in $\rho$-components of singly-anchored steps.

In the following definition, $\#(\mathfrak{l})$ is shorthand for the multiplicity $\#(G)$ of the corresponding guard $G^{\mathfrak{l}}$ in $P$.

**Definition 7.3.3 (Contradicting Strategies)** *Let $P$ be a process.*

*Two strategies $\rho_1$ and $\rho_2$* contradict *with respect to $P$ if there are two matching sequences of steps $\tilde{\pi}_1$ and $\tilde{\pi}_2$ such that one of the two following condition is satisfied:*

- *There are two contradicting sum guards $\mathfrak{l}_1$ and $\mathfrak{l}_2$ such that for both $i$, $\rho_i$ contains $\tilde{\pi}_i. \mathfrak{l}_i$.*
- *There are two steps $(\mathfrak{l}|\rho'_1)$ and $(\mathfrak{l}|\rho'_2)$ such that $\#(\mathfrak{l}) \neq \omega$ and $\rho_1$ doesn't match $\rho_2$, and, for both $i$, $\rho_i$ contains $\tilde{\pi}_i. (\mathfrak{l}|\rho'_i)$.*

Remember (Definition 7.1.4) that a strategy $\rho = (\mathfrak{l}|\rho_0)\lightning(\mathfrak{l}|\rho_1). \rho'$ doesn't contain $(\mathfrak{l}|\rho_0)$ but does contain $(\mathfrak{l}|\rho_1). \rho'$. So (assuming $\rho_0$ and $\rho_1$ don't match) $\rho$ contradicts $(\mathfrak{l}|\rho_0)$ but not $(\mathfrak{l}|\rho_1)$, as $\rho_1$ is $\mathfrak{l}$'s actual communication partner, although the strategy was "planning" to use $\rho_0$.

**Definition 7.3.4 (Choice Set)** *Let $P$ be an annotated process. A* choice set *for $P$ is a finite set of runnable liveness strategies (with or without a final step) such that no two strategies in the set contradict each other and that includes all sub-strategies of its elements.*

In particular, no strategy in a choice set may contradict itself, for instance by attempting to make $a$ and $\bar{a}$ communicate in $t.a + u.\bar{a}$, or using a linear channel more than once. Note that, just like some processes may have infinitely many liveness strategies in the presence of recursion, a process may have infinitely many choice sets.

An annotated behavioural statement is *complete* if it contains $\vee$-terms for every possible choice set, in other words if it is prepared to deal with any conceivable interference.

**Definition 7.3.5 (Completeness)** *An annotated behavioural statement $\Phi \cong \bigvee_{i \in I} \Phi_i$ is* complete *with respect to $P$ if, for every choice set $\tilde{\rho}_C$ there is $\hat{\imath} \in I$ such that no strategy appearing in $\Phi_{\hat{\imath}}$ contradicts any in $\tilde{\rho}_C$.*

Note that this is a very different concept to *channel type* completeness (Definition 4.2.7, page 35).

## 7.4 Annotated Labelled Transition System

We now lift the labelled transition system on typed processes to a labelled transition system on *annotated* typed processes.

When a transition on an annotated process brings an event closer to top-level, that event is replaced by an "extended event" — See grammar on page 73. Essentially, it is an liveness strategy where a step $(\mathfrak{l}_L|\mathfrak{l}_R)$ is abbreviated to $\mathfrak{l}_R$ — recording communication partners of prefixes that have already been consumed. This permits knowing the history of a process, which in turn is required in order to apply a strategy in the presence of interference. The following operator records one step of a strategy into a process:

**Definition 7.4.1 (Strategy Marking Operator)**
Marking *an annotated process $P$ with an event $\mathfrak{l}$, written* $\mathsf{mark}_{\mathfrak{l}}(P)$*, produces the annotated process inductively defined as follows:*

- $\mathsf{mark}_{\mathfrak{l}}(l) \overset{\text{def}}{=} \mathfrak{l}.\, l$
- $\mathsf{mark}_{\mathfrak{l}}(\mathfrak{l}_1.\,\mathfrak{l}_2) \overset{\text{def}}{=} \mathfrak{l}_1.\,\mathsf{mark}_{\mathfrak{l}}(\mathfrak{l}_2)$
- $\mathsf{mark}_{\mathfrak{l}}(G^{\mathfrak{l}'}.P) \overset{\text{def}}{=} G^{\mathsf{mark}_{\mathfrak{l}}(\mathfrak{l}')}.\mathsf{mark}_{\mathfrak{l}}(P)$
- $\mathsf{mark}_{\mathfrak{l}}(P_1|P_2) \overset{\text{def}}{=} \mathsf{mark}_{\mathfrak{l}}(P_1) \,|\, \mathsf{mark}_{\mathfrak{l}}(P_2)$
- $\mathsf{mark}_{\mathfrak{l}}(P_1{+}P_2) \overset{\text{def}}{=} \mathsf{mark}_{\mathfrak{l}}(P_1) + \mathsf{mark}_{\mathfrak{l}}(P_2)$
- $\mathsf{mark}_{\mathfrak{l}}((\boldsymbol{\nu}\mathfrak{a})\,P) \overset{\text{def}}{=} (\boldsymbol{\nu}\mathsf{mark}_{\mathfrak{l}}(\mathfrak{a}))\,(\mathsf{mark}_{\mathfrak{l}}(P)\{^{\mathsf{mark}_{\mathfrak{l}}(\mathfrak{a})}/_{\mathfrak{a}}\})$
- $\mathsf{mark}_{\mathfrak{l}}(\mathbf{0}) \overset{\text{def}}{=} \mathbf{0}$

For instance marking $a^{l}.P$ with $l_1$ returns $a^{l_1\cdot\, l}.\mathsf{mark}_{l_1}(P)$, and then marking that process with $l_2$ returns $a^{l_1\cdot\, l_2\cdot\, l}.\mathsf{mark}_{l_2}(\mathsf{mark}_{l_1}(P))$. Note how the operator always inserts a step just before the final one.

Based on the above marking operator we may now define the labelled transition system on annotated processes. Instead of the usual $P \xrightarrow{\mu} P'$ notation we write $P \xrightarrow{\mu,(\mathfrak{l}_l|\mathfrak{l}_r)} P'$ where $\mathfrak{l}_l$ indicates the strategy step corresponding to this transition (basically, which event it brings to top-level), and $\mathfrak{l}_r$ where the communication partner is found. In a $\tau$-reduction $P \xrightarrow{\tau,(\mathfrak{l}_i|\mathfrak{l}_o)} P'$, $\mathfrak{l}_i$ and $\mathfrak{l}_o$ indicate respectively the input and output prefixes that are communicating.

$$\frac{-}{\overline{a}\langle\tilde{x}\rangle^{\mathfrak{l}}.P \xrightarrow{\overline{a}\langle\tilde{x}\rangle,(\mathfrak{l}|\mathfrak{l}')} \mathsf{mark}_{\mathfrak{l}'}(P)}\ \text{(A-Out)}$$

$$\frac{-}{a(\tilde{y})^{\mathfrak{l}}.P \xrightarrow{a(\tilde{x}),(\mathfrak{l}|\mathfrak{l}')} \mathsf{mark}_{\mathfrak{l}'}(P)\{^{\tilde{x}}/_{\tilde{y}}\}}\ \text{(A-Inp)}$$

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\tilde{z}:\tilde{\sigma})\,\overline{a}\langle\tilde{x}\rangle,(\mathfrak{l}_o|\mathfrak{l}_i)} P' \quad Q \xrightarrow{a(\tilde{x}),(\mathfrak{l}_i|\mathfrak{l}_o)} Q'}{\begin{array}{c} P\,|\,Q \xrightarrow{\tau,(\mathfrak{l}_i|\mathfrak{l}_o)} (\boldsymbol{\nu}\tilde{z}:\tilde{\sigma})\,(P'\,|\,Q') \\ Q\,|\,P \xrightarrow{\tau,(\mathfrak{l}_i|\mathfrak{l}_o)} (\boldsymbol{\nu}\tilde{z}:\tilde{\sigma})\,(Q'\,|\,P') \end{array}}\ \text{(A-Com)}$$

Rules (A-Open), (A-Rep), (A-New), (A-Par), (A-Sum) and (A-Cong) are identical to the corresponding ones in Table 2.2 on page 16 except that they additionally carry $\mathfrak{l}$ components on the transition label without modification.

Using this labelled transition system, bringing an event $l$ to top-level transforms it into $\mathfrak{l}.\, l$, where $\mathfrak{l}$ is the strategy used for that.

As event annotations in processes change, liveness strategies need to be updated accordingly:

**Definition 7.4.2 (Strategy Transition Operator)** *Let $\rho$ be a strategy and $\pi$ an event pair $(\mathfrak{l}_1|\mathfrak{l}_2)$ where $\mathfrak{l}_2$ may be $\bullet$.*

*Then $\rho \wr \pi$ is the liveness strategy obtained as follows (the word "otherwise" is used in the sense "if none of the previous rules apply").*

- *If $\rho$ and $\pi$ contradict then $\rho \wr \pi = \bot$.*
- $\mathfrak{l} \wr \pi \overset{\text{def}}{=} \mathfrak{l}$ *otherwise.*
- *If one of $\pi.\, \rho_0$ and $\bar{\pi}.\, \rho_0$ matches $\pi_0.\, \rho_0$ then $(\pi_0.\, \rho_0) \wr \pi \overset{\text{def}}{=} \mathsf{mark}_{\mathfrak{l}_2}(\rho_0) \wr \pi$.*
- $((\mathfrak{l}_0|\rho_0).\, \rho_1) \wr \pi \overset{\text{def}}{=} (\mathfrak{l}_0|\rho_0 \wr \pi).\, (\rho_2 \wr \pi)$ *otherwise.*
- *$\pi_0\, [\, \tilde{q}\, ] \wr \pi = \bot$ if $\pi$ or $\bar{\pi}$ matches $\pi_0$.*
- $(\bullet|\rho_0)\, [\, \tilde{q}\, ] \wr \pi = (\bullet|\rho_0 \wr \pi)\, [\, \tilde{q}\, ]$ *otherwise.*
- $(\tilde{\pi} \lightning (\pi)\delta) \wr \pi \overset{\text{def}}{=} (\tilde{\pi} \lightning (\pi)\delta) \wr \bar{\pi} \overset{\text{def}}{=} (\pi\delta) \wr \pi$.
- $\big(\tilde{\pi} \lightning (\tilde{\pi}')\delta\big) \wr \pi \overset{\text{def}}{=} (\tilde{\pi} \wr \pi) \lightning \big((\tilde{\pi}')\delta \wr \pi\big)$ *otherwise.*

*with the following extension of the* $\mathsf{mark}$ *operator from Definition 7.4.1:*

- $\mathsf{mark}_{\mathfrak{l}}((\mathfrak{l}_0|\rho_0)) \overset{\text{def}}{=} (\mathsf{mark}_{\mathfrak{l}}(\mathfrak{l}_0)|\rho_0)$.
- $\mathsf{mark}_{\mathfrak{l}}(\tilde{\pi} \lightning (\rho_0)\delta) \overset{\text{def}}{=} \mathsf{mark}_{\mathfrak{l}}(\tilde{\pi}) \lightning (\rho_0)\delta$

*That operator is lifted to behavioural statements: $\Phi \mapsto \Phi \wr \pi$ is a logical homomorphism such that*

- *If $\Phi \cong \top$ then $\Phi \wr \pi \overset{\text{def}}{=} \top$.*
- *If $\rho \wr \pi = \bot$ then*
  - $(s_k \lhd \varepsilon \colon \rho) \wr \pi \overset{\text{def}}{=} \bot$
  - $(p_{\mathbf{R}} \lhd \varepsilon \colon \rho.\, \phi) \wr \pi \overset{\text{def}}{=} \top$
- *otherwise,*
  - $(s_k \lhd \varepsilon \colon \rho) \wr \pi \overset{\text{def}}{=} s_k \lhd \varepsilon \colon (\rho \wr \pi)$
  - $(p_{\mathbf{R}} \lhd \varepsilon \colon \rho.\, \phi) \wr \pi \overset{\text{def}}{=} p_{\mathbf{R}} \lhd \varepsilon \colon (\rho \wr \pi).\, (\phi \wr \pi)$ *(where $\phi \wr \pi$ follows the same rules as $\Phi \wr \pi$, without the $\varepsilon$)*
- $\Phi \wr \pi \overset{\text{def}}{=} \Phi$ *when no other rules apply.*

*On process types,* $(\Sigma; \Phi_{\mathrm{L}} \blacktriangleleft \Phi_{\mathrm{E}}) \wr \pi \overset{\mathrm{def}}{=} (\sigma; \Phi_{\mathrm{L}} \wr \pi \blacktriangleleft \Phi_{\mathrm{E}})$.

Transition on annotated typed processes are defined similarly to those on typed processes in Definition 3.11.4:

**Definition 7.4.3 (Typed Labelled Transition System)**
*The transition operator* $\Gamma \wr \mu$ *on annotated process types modifies the type precisely as in Definition 5.1.6 on page 50.*

*An annotated typed process has a transition*

$$(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$$

*if there is* $\pi$ *such that* $P \xrightarrow{\mu,\pi} P'$ *and* $(\Gamma \wr \mu) \wr \pi = \Gamma'$*. If* $\pi = (\mathfrak{l}_l | \mathfrak{l}_r)$ *and* $\mu \neq \tau$ *then* $\mathfrak{l}_r$ *must be* $\bullet$*.*

Note that $\sigma[\tilde{x}]$ and $\overline{\sigma}[\tilde{x}]$ used in Definition 5.1.6 contain no strategies, so $\Gamma \wr \mu$ yields a "mixed" process type that contains strategy annotations on some statements but not all. As we will see, the weakening constraint from Definition 5.2.6 drops precisely those statements that do not have strategies.

The following lemma is easily shown by dropping all strategy annotations on transition labels and processes and noting that it reduces to the LTS seen in Section 3.6. The reciprocal is obtained by annotating transitions with strategies obtained from the process, and inductively constructing the labelled transition sequence as indicated by the rules (A-Inp) and (A-Out).

**Lemma 7.4.4 (LTS Equivalence)** *Let* $(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'; P')$ *be a transition sequence on annotated typed processes. Then* $\mathrm{ran}(\Gamma; P) \xrightarrow{\tilde{\mu}} \mathrm{ran}(\Gamma'; P')$*.*

*Let* $\mathrm{ran}(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'; P')$ *be a transition sequence on* non-*annotated typed processes. Then there is exactly one* $(\Gamma'_0; P'_0)$ *such that* $(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'_0; P'_0)$ *and* $\mathrm{ran}(\Gamma'_0; P'_0) = P'$*.*

The following lemma tells how strategy subjects evolve when the process goes through a transition. It serves as a base to proving safety of runnability.

**Lemma 7.4.5 (Subject Transitions)** *Let* $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ *be a transition on annotated typed process and let* $\pi$ *be the strategy step used to prove it using Definition 7.4.3, and let* $\mathfrak{p}$ *an extended port. Then there is an extended port* $\mathfrak{p}'$ *such that, for any runnable strategy* $\rho$ *such that* $\rho \wr \pi \neq \bot$*,* $\mathsf{sub}_P(\rho) = \mathfrak{p}$ *implies* $\mathsf{sub}_{P'}(\rho \wr \pi) = \mathfrak{p}'$*, and* $\mathsf{sub}_P(\rho) = \overline{\mathfrak{p}}$ *implies* $\mathsf{sub}_{P'}(\rho \wr \pi) = \overline{\mathfrak{p}'}$*, and similarly for* $\mathsf{trg}_{k,P}$*.*

See Section A.4.1 for the proof.

The previous lemma implies (proved as part of Lemma 7.4.8) that runnable strategies and consistent types remain runnable and consistent when the process evolves.

The following one is in some sense a reciprocal, in that if $P \xrightarrow{\tilde{\mu}} Q$, for any $Q$-runnable strategy $\rho'$ there is a corresponding $P$-runnable strategy $\rho$ such that $\rho \wr \tilde{\pi} = \rho'$ (where $\tilde{\pi}$ is the sequence of steps corresponding to $\tilde{\mu}$), which in turn guarantees that a complete type remains complete when the process evolves.

**Lemma 7.4.6 (Completeness of Strategies)** *Let* $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ *be a transition that, if it is an input, has only fresh and distinct objects. Let* $\mathfrak{p}'$ *be an extended port. Then there is an extended port* $\mathfrak{p}$ *such that:*

*For all runnable strategies* $\rho'$ *such that* $\mathsf{sub}_{P'}(\rho') = \mathfrak{p}'$ *there is a strategy* $\rho$ *that satisfies the guarding and top-levelness constraints of Definition 7.2.3, such that* $\mathsf{sub}_P(\rho) = \mathfrak{p}$ *and* $\rho \wr \pi = \rho'$*.*

*The same properties hold substituting* $\mathfrak{p}'$ *with* $\overline{\mathfrak{p}'}$ *and* $\mathfrak{p}$ *with* $\bar{\mathfrak{p}}$ *(i.e. the* $\mathfrak{p}' \mapsto \mathfrak{p}$ *transformation commutes with the complement operator).*

See Section A.4.2 for the proof.

The *weight* of a strategy (over-) estimates how many transitions are required to bring its final step to top-level and is an essential component of proving liveness.

**Definition 7.4.7 (Weight of a Strategy)** *The* weight $\mathsf{wt}(\rho)$ *of a strategy* $\rho$ *is defined inductively:*

- $\mathsf{wt}(\mathfrak{l}) \overset{\text{def}}{=} \mathsf{wt}(\bullet) \overset{\text{def}}{=} 0$
- $\mathsf{wt}((\bullet|\rho)\,[\,p\,]) \overset{\text{def}}{=} \mathsf{wt}((\mathfrak{l}|\rho)) \overset{\text{def}}{=} \mathsf{wt}((\mathfrak{l}|\rho]) \overset{\text{def}}{=} \mathsf{wt}(\rho) + 1$
- $\mathsf{wt}(\pi.\,\rho) \overset{\text{def}}{=} \mathsf{wt}(\pi) + \mathsf{wt}(\rho)$
- $\mathsf{wt}(\tilde{\pi}_1 \lightning (\tilde{\pi}_2)\delta) \overset{\text{def}}{=} \mathsf{wt}(\tilde{\pi}_1) + \mathsf{wt}(\tilde{\pi}_2\delta) - \mathsf{wt}(\tilde{\pi}_2)$

"Elementary" in the following definition refers to the image of relation $\searrow$. See Definition 4.3.3 on page 39.

**Lemma 7.4.8 (Runnability Safety)** *Let* $(\Gamma; P)$ *be an annotated typed process that is consistent, complete and elementary.*

*Then for any transition* $(\Gamma; P) \xrightarrow{\mu}\searrow (\Gamma'; P')$ *such that* $\Gamma \preceq \Gamma'$*,* $(\Gamma'; P')$ *is consistent, complete and elementary as well, and* $\mathsf{wt}(\Gamma) \leq \mathsf{wt}(\Gamma')$*.*

See Section A.4.3 for the proof.

A key component of proving correctness of an annotated process type is the following lemma, that effectively connects process structure (liveness strategies) and process behaviour (transition sequences).

**Lemma 7.4.9 (Strategy Application)** *Let* $(\Gamma; P)$ *be a consistent, complete and elementary annotated typed process, and let* $\tilde{\rho}$ *be a choice set.*

*Then either* $(\Gamma; P)$ *is immediately correct or there is a transition* $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ *such that* $\mathsf{wt}(\Gamma') < \mathsf{wt}(\Gamma)$*.*

See Section A.4.4 for the proof.

Soundness of consistency and completeness follows as a simple corollary.

**Corollary 7.4.10 (Completeness and Correctness)** *Let* $(\Gamma; P)$ *be a complete and consistent annotated typed process. Then* $\mathrm{ran}(\Gamma; P)$ *satisfies definition 5.2.6 for the special case where* $\tilde{\mu}_0$ *is empty.*

See Section A.4.5 for the proof.

## 7.5 Annotated Type System

We now extend various process type operators to work with annotated process types and gradually build strategies as a process is being run through the type system:

1. The relation $\hookrightarrow$, when used to reduce dependency chains, has to combine the corresponding strategies.
2. The (E-PRE) rule constructs base strategies for the subject responsiveness and liveness properties, remote behaviour, and adds a transition at the beginning of all activeness strategies from the continuation.

The following definition tells how to reduce a $s_k \lhd p_{k'} \lhd \varepsilon_p$ dependency chain with $\{k, k'\} \subseteq \mathcal{E}$ :

**Definition 7.5.1 (Liveness-Liveness Reduction)** *The $\hookrightarrow$ relation is modified as follows for annotated process types, in the context of a process $P$:*
*Let $\Xi = (p_{k'} \lhd \varepsilon_p : \rho_p) \wedge (s_k \lhd (p_{k'} \wedge \varepsilon_s) : \rho_s)$. Then*

$$\Xi \hookrightarrow \Xi \ \wedge \ s_k \lhd (\varepsilon_p \wedge \varepsilon_s) : \rho'_s$$

*$\rho'_s$ is obtained from $\rho_s$ by replacing as many sub-strategies as possible using the following rules, that each assume $\mathsf{sub}_P(\tilde{\pi}.\, l) = \bar{p}$.*

$$\tilde{\pi}.\,(l|\bullet).\,\rho \mapsto \tilde{\pi}.\,(l|\rho_p).\,\rho \tag{7.5}$$

$$\tilde{\pi}.\,(l|\bullet).\,\rho \mapsto \tilde{\pi}.\,(l|\rho_p).\,\rho \lightning \big(\tilde{\pi}.\,(l|\bullet)\big).\,\rho \tag{7.6}$$

$$(\bullet|\tilde{\pi}.\,\mathfrak{l})\,[\,r\,] \mapsto (\rho_r|\tilde{\pi}.\,\mathfrak{l}) \lightning (\bullet|\tilde{\pi}.\,\mathfrak{l})\,[\,r\,] \tag{7.7}$$

To reduce chains such as $s_k \lhd p_{k'} \lhd \varepsilon_p$ with $k \in \mathcal{E}$ and $k' \in \mathcal{U}$, one needs to convert a responsiveness strategy $\phi$ on parameter names into an liveness strategy, applying parameter instantiation to strategies:

**Definition 7.5.2 (Strategy Instantiation)** *Let $\phi$ be an annotated responsiveness strategy and $s$ a sum of parameter ports ($n$ or $\bar{n}$). Then* instantiating *$\phi$'s port(s) $s$, written $\phi\,[\,s\,]$ is the logical homomorphism returning behavioural statements whose atoms are liveness strategies:*

- *$(s'_k : \rho)\,[\,s\,] = \rho$ when $s = s'$*
- *$\phi\,[\,s\,] = \top$ when no other rules apply.*

Extracting an liveness strategy from a responsiveness strategy replaces the "unspecified" communication partner "$\bullet$" and parameter number "$[\,s\,]$" by an actual communication partner $\rho_p$ and an instantiation of its responsiveness strategy $\phi\,[\,s\,]$.

**Definition 7.5.3 (Responsiveness-Liveness Reduction)** *Let $\Xi = p_{\mathbf{R}} \lhd \varepsilon_p : \rho_p.\,\phi$ be an annotated behavioural statement for a process $P$ and $k$ an existential property. Then*

$$\Xi \wedge (s_k \lhd (p_{\mathbf{R}} \wedge \varepsilon_s) : \rho_s) \hookrightarrow \Xi \wedge \big((s_k \lhd (p_{\mathbf{R}} \wedge \varepsilon_s) : \rho_s) \vee (s_k \lhd (\varepsilon_p \wedge \varepsilon_s) : \rho'_s)\big) \tag{7.8}$$

*where $\rho'_s$ is obtained from $\rho_s$ by repeatedly applying the following transformation on sub-strategies:*

$$(\bullet|\rho_1)\,[\,s'\,] \mapsto (\bullet|\rho_1)\lightning(\rho_p|\rho_1).\,\phi\,[\,s'\,]$$

The following definition tells how the above reduction rules descend into responsiveness strategies. We use the logical homomorphism $\phi \mapsto (\rho|\bullet).\,\phi$ that maps $p_k \lhd \varepsilon\colon \rho'$ to $p_k \lhd \varepsilon\colon (\rho|\bullet).\,\rho'$ and $p_{\mathbf{R}} \lhd \varepsilon\colon \rho'.\,\phi$ to $p_{\mathbf{R}} \lhd \varepsilon\colon ((\rho|\bullet).\,\rho').\,\phi$.

**Definition 7.5.4 (Responsiveness Reduction)** *Let $\Xi$ be an annotated dependency statement and $\phi$ a responsiveness strategy such that $\Xi \wedge (\rho|\bullet).\,\phi \hookrightarrow \Xi \wedge (\rho'|\bullet).\,\phi'$ for some $\phi'$. Then*

$$\Xi \wedge (p_{\mathbf{R}} \lhd \varepsilon\colon \rho.\,\phi) \hookrightarrow \Xi \wedge (p_{\mathbf{R}} \lhd \varepsilon'\colon \rho'.\,\phi')$$

*where $\varepsilon'$ is obtained as in Definition 5.1.3.*

Gathering the above definitions together we obtain the annotated counterpart to Definition 5.1.3 on page 49. There are fewer cases because annotated process types only contain dependency statements on the local side.

**Definition 7.5.5 (Annotated Dependency Reduction)** *The reduction relation $\hookrightarrow$ on annotated behavioural statements is a partial order relation satisfying*

- *The reductions as given in Definitions 7.5.1, 7.5.3 and 7.5.4.*
- *$\Phi \hookrightarrow \Phi'$ implies $(C[\Phi] \blacktriangleleft \Phi_{\mathrm{E}}) \hookrightarrow (C[\Phi'] \blacktriangleleft \Phi_{\mathrm{E}})$ for any local context $C[\cdot]$.*

The above relation preserves consistency:

**Lemma 7.5.6 (Reduction Preserves Consistency)** *Let $\Phi$ be a consistent annotated behavioural statement for a process $P$, and $\Phi \hookrightarrow \Phi'$. Then $\Phi'$ is consistent as well.*

And so does composition:

**Lemma 7.5.7 (Composition Preserves Consistency)** *Let $\Gamma_1, \Gamma_2$ be annotated process types consistent for a process $P$. Then their composition $\Gamma_1 \odot \Gamma_2$ is consistent for $P$ as well.*

The proofs are in Section A.4.6.

When the $\hookrightarrow$ relation replaces some strategy $\rho_0$ by $\rho$, $\rho_0$ is a *precursor* of $\rho$. The two points in the list below respectively model transformations done by Definitions 7.5.1 and 7.5.3 on page 89.

**Definition 7.5.8 (Strategy Precursor)** *A liveness strategy $\rho_0$ is said a* precursor *of a strategy $\rho$ for some process $P$ if $\rho$ can be obtained from $\rho_0$ by applying zero, one or more times the following transformations, while preserving the $\mathsf{sub}_P(\rho_0) = \mathsf{sub}_P(\rho)$ equality.*

- *replacing some $\bullet$ by liveness strategies,*
- *replacing a sub-strategy $(\bullet|\rho_0)\,[\,q\,]$ by $\tilde{\pi}.\,(\wr|\rho_0).\,\rho'$*

As far as completeness is concerned, dependency reduction transforms an incomplete type into a complete one, as long as responsiveness of every port is available, and every strategy has a matching precursor with $\bullet$-steps that can be used for performing dependency reduction. This is formalised as follows.

**Definition 7.5.9 (Pre-Completeness)** *Let $P$ a process and $\tilde{\rho}$ be a choice set. An annotated behavioural statement $\Phi$ is said to be* pre-complete *for $P$ with respect to $\tilde{\rho}$ if:*

- *No liveness strategy in $\Phi$ is self-contradicting.*
- *for any runnable liveness strategy $\rho$ not contradicting $\tilde{\rho}$ and such that $\mathsf{sub}_P(\rho)$ is a free port $p$, $\Phi$ contains a statement $p_{\mathbf{R}} \lhd \varepsilon \colon \rho_0 .\, \phi$ with $\rho_0$ being a precursor of $\rho$.*
- *for every annotated liveness statement $p_k \lhd \varepsilon \colon \rho_2$ contained in $\Phi$, for every runnable precursor $\rho_1$ of $\rho_2$, there is a precursor $\rho_0$ of $\rho_1$ such that $\Phi$ contains a statement $p_k \lhd \varepsilon' \colon \rho_2$ for some $\varepsilon'$.*

*An annotated behavioural statement $\bigvee_i \Phi_i$ is* pre-complete *for $P$ if, for all choice sets $\tilde{\rho}$ there is $i$ such that $\Phi_i$ is pre-complete for $P$ with respect to $\tilde{\rho}$.*

We conjecture completeness implies pre-completeness but it is not needed for the soundness proof.

As we will see in the annotated type system soundness proof below, if $\Phi_1$ and $\Phi_2$ are pre-complete for two processes $P_1$ and $P_2$ then their composition $\Phi_1 \odot \Phi_2$ is pre-complete as well. Recall that composition of *behavioural statements*, from Definition 5.1.1 on page 48, does not perform a closure, so there is no similar result for completeness, but composing and then performing the the closure of two complete process types gives a complete type:

**Lemma 7.5.10 (Closure Completes)** *The closure of a consistent and pre-complete type $\Gamma$ for a process $P$ is complete for $P$.*

The proof is in Section A.4.7

We introduce a few notations used by the annotated type system rules.

Having $\mathrm{n}(p) = a$, "$\xi_p$" is $\xi_{\mathrm{I}}$ if $p = a$, and $\xi_{\mathrm{O}}$ if $p = \bar{a}$ (This notation is used in the third statement of the resulting type in (R-PRE)). That behavioural statement is then applied the logical homomorphism $\varepsilon \colon \bullet$ that annotates every resource with the vacuous strategy $\bullet$ (for instance $(1_{\mathbf{A}} \wedge 2_{\mathbf{A}}) \colon \bullet = (1_{\mathbf{A}} \colon \bullet) \wedge (2_{\mathbf{A}} \colon \bullet)$).

*Strategy prefixing* $(l|\bullet).\, \Gamma$ applies a logical homomorphism such that $\pi .\, (s_k \lhd \varepsilon \colon \rho') \overset{\text{def}}{=} s_k \lhd \varepsilon \colon (\pi .\, \rho')$ and $\pi .\, \Gamma \overset{\text{def}}{=} \Gamma$ when no other rules apply. This notation is used in the last statements of the conclusion in (R-PRE).

Finally, *Annotated Parameter Instantiation* $\sigma[\tilde{x}]_l$ is like $\sigma[\tilde{x}]$ but replacing any $(x_i)_k \lhd \varepsilon$ (resp., $(\overline{x_i})_k \lhd \varepsilon$) by $(x_i)_k \lhd \varepsilon \colon (\bullet|l)\,[\,i\,]$ (resp., $(\overline{x_i})_k \lhd \varepsilon \colon (\bullet|l)[\,\bar{\imath}\,]$). Regarding sums, $\left(\sum_i x_i\right)_{\mathbf{A}} \lhd \varepsilon$ becomes $\left(\sum_i x_i\right)_{\mathbf{A}} \lhd \varepsilon \colon (\bullet|l)\,[\sum_i i]$

**Definition 7.5.11 (Annotated Type System)** *The Annotated Type System works like the one in Section 5.3 but constructs strategies for each dependency statement, using the rules from Table 7.2 (that only contains the rules that are different from the ones in Table 5.1).*

$$\frac{\forall i : (\Sigma_i; \Phi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash'_{\mathcal{K}} {G_i}^{l_i}.P_i \qquad \Xi_{\mathrm{E}} \preceq \bigwedge_i \Xi_{\mathrm{E}i}}{\left(\bigwedge_i \Sigma_i; \bigwedge_{k\in\mathcal{K}} \mathsf{sum}_k(\{p_i\}_i, \Xi_{\mathrm{E}}) \colon \sum_i l_i \wedge \bigvee_i \Phi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}}\right) \vdash'_{\mathcal{K}} \sum_i {G_i}^{l_i}.P_i}\ (\text{R-Sum})$$

$$\frac{\Gamma \vdash'_{\mathcal{K}} P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x} \qquad \sigma = (\tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}})}{\begin{array}{r} \left(p : \sigma;\ \blacktriangleleft p^m \wedge \bar{p}^{m'}\right) \quad \odot \\ (; p^{\#(G)} \blacktriangleleft) \quad \odot \\ !_{\text{if } \#(G) = \omega}\ (\boldsymbol{\nu}\mathrm{bn}(G)) \Big(\ (l|\bullet).\, \Gamma \lhd \mathsf{dep}_{\mathcal{K}}(G) \quad \odot \\ \bar{\sigma}[\tilde{x}]_l \lhd (\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}}) \quad \odot \\ (; \bigwedge_{k\in\mathcal{K}} \mathsf{prop}_k(\sigma, G, m, m') \colon l \blacktriangleleft) \quad \odot \\ (; p_{\mathbf{R}} \lhd \sigma[\tilde{x}] \colon l.\, (\xi_p \colon \bullet) \blacktriangleleft) \quad \Big) \vdash'_{\mathcal{K}} G^l.P \end{array}}\ (\text{R-Pre})$$

Table 7.2: Annotated Rules

The following lemma is shown by a trivial structural inductive proof, as the behaviour of operators with respect to dependencies was not modified:

**Lemma 7.5.12 (Type System Equivalence)** *Let* $(\Gamma; P)$ *be a typed process such that* $\Gamma \vdash_{\mathcal{K}} P$. *then there is an annotated typed process* $(\Gamma'; P')$ *such that* $\Gamma' \vdash'_{\mathcal{K}} P'$ *and* $\mathrm{ran}(\Gamma'; P') = (\Gamma; P)$.

Given $P$ and the typing $\Gamma \vdash_{\mathcal{K}} P$, the annotated form $P'$ is done by replacing every guarded process $G.P$ by $G^l.P$, where $l$ is the event that was used in the rule (E-Pre) for that prefix in the derivation for $\Gamma \vdash_{\mathcal{K}} P$.

**Lemma 7.5.13 (Annotated Type System Soundness)**
*Let* $(\Gamma; P)$ *be an annotated typed process such that* $\Gamma \vdash'_{\mathcal{K}} P$. *Then* $(\Gamma; P)$ *is consistent and complete.*

The proof is in Section A.4.8.

The framework introduced until now does not deal with choice guarded by a replicated prefix (as in $!\, a(x).(P+Q)$). For instance no runnable strategy can model the sequence

$$P = !\, u.(\bar{a}+a.\bar{s}) \xrightarrow{u} P \,|\, (\bar{a}+a.\bar{s}) \xrightarrow{u} P \,|\, (\bar{a}+a.\bar{s}) \,|\, (\bar{a}+a.\bar{s}) \xrightarrow{\tau} P \,|\, \bar{s}$$

We reserve such an extension for future work and for the time being will merely sketch a proof that the $!$ operator (in particular the dependency reductions it entails) preserves completeness.

Let $(\Gamma_0; P_0)$ be an annotated typed process with $\Gamma_0 \vdash'_{\mathcal{K}} P_0$. By induction, $\Gamma_0$ is consistent and complete for $P_0$. We show that $(\Gamma'; G^l.P_0)$, where $\#(G) = \omega$ and $\Gamma'$ is obtained from $\Gamma_0$ following (R-Par), is consistent and complete as well. Let $\Gamma$ be the type under replication, i.e. the composition of continuation, remote parameters and responsiveness. Remember (Definition 4.4.7 on page 43) that $!\,\Gamma \overset{\text{def}}{=} \Gamma \odot \Gamma \odot \cdots \odot \Gamma$ with as many instances as there are $\vee$-terms in $\Gamma$'s local component (multiplied by two to make sure all multiplicities are $\star$ but we aren't concerned about multiplicities here). By $\Gamma$'s completeness, that number

$n$ of terms is the number of classes of possible choice sets (where two choice sets are in the same class if the same $\vee$-term is complete with respect to both of them). Conversely, to any choice set can be associated a number between 1 and $n$.

Now consider a transition sequence $G^l.P_0 = P \xrightarrow{\tilde{\mu}} P'$. $P'$ can be decomposed into a product $P \mid (\boldsymbol{\nu}\tilde{z})\,\big(P_1 \,|\, P_2 \,|\, \ldots \,|\, P_m\big)$ where $m$ is the number of time the $G$-prefix got invoked and, for all $i$, $P \xrightarrow{\tilde{\mu}_i} P|P_i$, for some $\tilde{\mu}_i$. By LTS equivalence, that sequence $\tilde{\mu}_i$ can be converted into a sequence of steps $\tilde{\pi}_i$. They are necessarily non-contradictory, as they correspond to an actual transition sequence, therefore form a choice set and have a matching $\vee$-term $n_i$ in $\Gamma$.

Replace every event $\mathfrak{l}$ occurring in processes in the sequence $P \xrightarrow{\tilde{\mu}} P'$ by a pair $(\mathfrak{l}, n_i)$ where $i$ is the process containing the event. In $\Gamma = \bigvee_i \Gamma_i$, similarly replace, in each $\Gamma_i$, every event $\mathfrak{l}$ (other than $l$ itself) by the pair $(\mathfrak{l}, i)$.

This extended framework guarantees the following property: all intermediate processes in the $P$-$P'$ sequence are of the form $P \mid (\boldsymbol{\nu}\tilde{x})\,(\hat{P}_1 \mid \hat{P}_2 \mid \ldots \mid \hat{P}_{m'})$ ($m'$ is not related in any way to $n$ or $m$, as $G$'s continuation $P_0$ may itself be a parallel composition of processes), such that if an event ($\mathfrak{l}$ or $(\mathfrak{l}, i)$) appears more than once, it is in two processes $\hat{P}_j$ and $\hat{P}_k$ with $\hat{P}_j = \hat{P}_k\{^{\tilde{x}}/_{\tilde{y}}\}$ where $\tilde{x}$ and $\tilde{y}$ are distinct names appearing only in $\hat{P}_j$ (respectively, $\hat{P}_k$).

As events paired with a number $i$ all perform the same choices by construction, there are no contradictory sequences and the closure of $\Gamma^{2n}$ is complete.

## 7.6 Overall Soundness Proof (Proposition 5.6.4)

We may now formulate the proof of the Soundness Proposition as a corollary of the previous lemma:

Let $(\Gamma; P)$ be a (non-annotated) typed process such that $\Gamma \vdash_{\mathcal{K}} P$.

Let an arbitrary transition sequence

$$(\Gamma; P) = (\Gamma_0; P_0) \xrightarrow{\tilde{\mu}_0}\!\searrow (\Gamma'_0; P'_0) \tag{7.9}$$

By Subject Reduction (Prop. 5.6.3), there is $\Gamma''_0 \preceq \Gamma'_0$ such that $\Gamma''_0 \vdash_{\mathcal{K}} P'_0$.

By the Type System Equivalence there is an annotated typed process $(\hat{\Gamma}'_0; \hat{P}'_0)$ such that $\hat{\Gamma}'_0 \vdash'_{\mathcal{K}} \hat{P}'_0$ and $\operatorname{ran}(\hat{\Gamma}'_0; \hat{P}'_0) = (\Gamma''_0; P'_0)$.

By the annotated type system soundness, $\hat{\Gamma}'_0$ is consistent and complete for $Q'$.

Let $(\hat{\Gamma}'_0; \hat{P}'_0) \xrightarrow{f} (\hat{\Gamma}_1; \hat{P}_1) \xrightarrow{\tilde{\mu}_1}\!\searrow \ldots$ be an arbitrary transition sequence where the $\tilde{\mu}_i$ satisfy the constraints given in Definition 5.2.6 and $f$ is constructed as given in the completeness soundness Lemma. By that same lemma, $(\hat{\Gamma}_n; \hat{P}_n)$ is immediately correct for some $n$.

By LTS equivalence that transition sequence can be translated into a sequence on non-annotated processes $(\Gamma''_0; P'_0) \xrightarrow{f} (\Gamma_1; P_1) \xrightarrow{\tilde{\mu}_1}\!\searrow \ldots$ where $(\Gamma_i; P_i) = \operatorname{ran}(\hat{\Gamma}_i; \hat{P}_i)$ and $(\Gamma''_i; P'_i) = \operatorname{ran}(\hat{\Gamma}'_i; \hat{P}'_i)$.

Annotation removal preserves immediate correctness so $(\Gamma_n; P_n)$ is immediately correct as well.

As $\Gamma''_0 \preceq \Gamma'_0$ there is a similar transition sequence starting from $(\Gamma'_0; P'_0)$ with equal processes and transitions. As weakening commutes with the transition operator the $n^{\text{th}}$ typed process in that sequence is a weakening of $(\Gamma_n; P_n)$ so it is immediately correct as well. Connecting that transition sequence with (7.9) gives a sequence matching the requirements of Definition 5.2.6. As this works for an arbitrary sequence we get $\Gamma \models P$.

## 7.7 Structural Analysis for Process-Level Properties

We conclude this section on structural analysis with a tool useful for translating a process-level universal behavioural property into a channel-level property.

Recall that, when introducing process types (Section 3.4) we considered the interface between a process and its environment as a special kind of channel (let's call that channel $\mathsf{proc}$). It therefore makes sense to continue this analogy by having process properties $\mathsf{proc}_k$ as a special case of channel properties $p_k$. Applying that reasoning backwards, a formal definition of a process behavioural property (such as termination, determinism, isolation, etc) can be translated into a channel-level property.

**Definition 7.7.1 (Process-Level Property)** *Let $\varphi$ be a semantic predicate on typed processes such that $\varphi(\varepsilon, (\Gamma; P))$ is true or false for any given dependency $\varepsilon$ and typed process $(\Gamma; P)$, and only using $P$ through its transition network.*

*Then the associated $\mathsf{good}_\varphi$ predicate is such that $\mathsf{good}_\varphi(p \triangleleft \varepsilon, (\Gamma; P))$, for an annotated typed process $(\Gamma; P)$, is true if the following holds.*

*Let $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ be an arbitrary transition with $\mathsf{sub}(\mu) = p$. Following Definition 7.4.3, let $P \xrightarrow{\mu,(l_i|l_o)} P'$ be the corresponding process transition, and $P \xrightarrow{\mu,(\hat{l}_i|\hat{l}_o)} \hat{P}'$ be s.t. $(\hat{l}_i|\hat{l}_o)$ is obtained from $(l_i|l_o)$ by replacing $\bullet$ with a fresh event $l_T$.*

*Then $\varphi(\varepsilon, (\Gamma'; P'))$ holds on the subset of the labelled transition from $P'$ that uses $l_T$ in the sequences.*

Similarly, we translate an elementary rule giving $\mathsf{proc}_k$'s dependencies in a process $P$ into a rule giving $a_k$'s dependencies in $a.P$. For this, the elementary rule needs to know the subject of the parent guard. We write $\mathsf{prop}_{p_k}(\sigma, G, m, m')$ for the $k$-elementary rule applied on guard $G$ whose guard has subject $p$. When typing $P$ as a sub-process of $a.P$ we use the notation $\Gamma \vdash^p_{\mathcal{K}} P$, and let $\Gamma \vdash_{\mathcal{K}} P$ stand for $\Gamma \vdash^{\mathsf{proc}}_{\mathcal{K}} P$.

**Definition 7.7.2 (Process-Level Elementary Rules)** *Let $\varphi_k$ be a function mapping tuples $(\sigma, G, m, m')$ to the dependencies $\varepsilon$ of $\mathsf{proc}_k$. Then the corresponding elementary guard rule $\mathsf{prop}_{p_k}$ is such that*

$$\mathsf{prop}_{p_k}(\sigma, G, m, m') \overset{\text{def}}{=} p_k \triangleleft \varphi_k(\sigma, G, m, m')$$

*Let $\varphi_k$ be a function mapping pairs $(\tilde{p}, \Xi)$ to the dependencies $\varepsilon$ of $\mathsf{proc}_k$. Then the corresponding elementary sum rule $\mathsf{sum}_{p_k}$ is such that*

$$\mathsf{sum}_{p_k}(\tilde{p}, \Xi) \overset{\text{def}}{=} p_k \triangleleft \varphi_k(\tilde{\pi}, \Xi)$$

# Chapter 8

# Applications

I will now describe a number of universal properties (briefly covered in the Introduction, Section 1.2) one may want to enforce in processes.

## 8.1 Isolation

A conversation between a client and a server is *isolated* if no third-party is able to obtain information about it (not even the fact that a connection has been established). For instance the process $!\, a(x).(\bar{s}|\bar{x}\langle t\rangle)$ replies to every request on $a$ with a reference to $t$, but also sends a signal to $s$ every time a request is sent, so that a third-party process listening on $s$ is notified every time a client connects to $a$, violating isolation.

The principle is simple: a process $\bar{a}$ is isolated if the receiver at $a$ is itself isolated, written $\mathsf{proc}_{\mathbf{I}} \lhd a_{\mathbf{I}}$. If $\bar{a}$ is not observable then, following Definition 5.1.5, that statement reduces to $\mathsf{proc}_{\mathbf{I}} \lhd \top \cong \mathsf{proc}_{\mathbf{I}}$, i.e. the process is isolated. Dependency reduction deals with signals sent from behind a prefix: for instance when typing a process $P = \bar{t} \,|\, t.Q$ (assume $t$ is linear), let $Q$'s isolation depend on $\varepsilon$. Then $\bar{t}$'s type is $\mathsf{proc}_{\mathbf{I}} \lhd t_{\mathbf{I}}$, and $t.Q$'s type is $(t_{\mathbf{I}} \lhd \varepsilon) \wedge (\mathsf{proc}_{\mathbf{I}} \lhd \bar{t}_{\mathbf{I}})$. Composing these two types reduces the chain $\mathsf{proc}_{\mathbf{I}} \lhd t_{\mathbf{I}} \lhd \varepsilon$ to $\mathsf{proc}_{\mathbf{I}} \lhd \varepsilon$.

In terms of behavioural statements, we use the notation $a_{\mathbf{I}}$ to mean that $a$ is *isolated*. Then, in $!\, a(x).P$, $a_{\mathbf{I}}$ will depend on isolation of every name free in $P$ (for example $a$ is isolated in the forwarder $!\, a(x).\bar{b}\langle x\rangle$ if $b$ is isolated, i.e. $a_{\mathbf{I}} \lhd b_{\mathbf{I}}$).

A port $p$ with a plain multiplicity "$\star$" can't be isolated because requests to it may be intercepted by a third party, so the prefix rule of the type system introduces a statement $p_{\mathbf{I}} \lhd \perp$ for every such port.

**Definition 8.1.1 (Isolation Semantics)** *The* $\mathsf{good}_{\mathbf{I}}$ *predicate is obtained following Definition 7.7.1 with* $\varphi(\varepsilon, (\Gamma; P))$ *being true if* $(\Gamma; P) \xrightarrow{\mu}$ *with* $\mu \neq \tau$ *implies* $\mathsf{sub}(\mu)_{\mathbf{I}} \succeq \varepsilon$.

Note how a process with no free name can't have non-$\tau$-transition and is necessarily isolated according to this definition.

The elementary rule is as expected:

**Definition 8.1.2 (Elementary Isolation Rule)** *The elementary guard rule for isolation* $\mathsf{prop}_{\mathbf{I}}$ *is obtained following Definition 7.7.2 with* $\varphi_{\mathbf{I}}(\sigma, G, m, m') = \mathsf{sub}(G)_{\mathbf{I}}$.

With isolation just like with most universal properties, self-dependencies are actually not harmful. For instance the live-locked process $(\boldsymbol{\nu} a)\,(!\,a.\bar{a}\,|\,\bar{a})$ has no free name and is therefore isolated. However, when typing this process, recursion produces the statement $a_{\mathbf{I}} \lhd a_{\mathbf{I}}$ which reduces to $a_{\mathbf{I}} \lhd \bot$. Composing with the statement $\mathsf{proc}_{\mathbf{I}} \lhd a_{\mathbf{I}}$ produced by the top-level $\bar{a}$-output we get the type $\mathsf{proc}_{\mathbf{I}} \lhd \bot$, i.e. the process is deemed *not* isolated. The delayed dependency extension (Section 5.5) with the elementary rule $\varphi_{\mathbf{I}}(\sigma, G, m, m') = \overline{\mathsf{sub}(G)_{\mathbf{I}}}^{+1}$, would solve this particular issue as $a_{\mathbf{I}} \lhd a_{\mathbf{I}}$ would reduce to $a_{\mathbf{I}} \lhd \top$, keeping $\mathsf{proc}_{\mathbf{I}} \lhd {a_{\mathbf{I}}}^{+1}$ unchanged when typing $!\,a(\bar{a}).\,|\,\bar{a}$, and reducing to $\mathsf{proc}_{\mathbf{I}} \lhd \top$, as wanted, when binding $a$.

**Lemma 8.1.3 (Isolation Soundness)** *The isolation instance of the universal type system satisfies the conditions given in Definition 4.4.1*

*Proof* Requirements 1 and 3 hold by construction. For number 2 we need to show that $q_{\mathbf{I}} \lhd \varepsilon \succeq \mathsf{proc}_{\mathbf{I}} \lhd \overline{\mathsf{sub}(G)_{\mathbf{I}}}$ implies $\big(((\Gamma; G) \xrightarrow{\mu}) \Rightarrow \overline{\mathsf{sub}(\mu)_{\mathbf{I}}} \succeq \varepsilon\big)$ where $\Gamma = \Big(p : \sigma;\ \blacktriangleleft p^m \wedge \bar{p}^{m'}\Big)$ and $p = \mathsf{sub}(G)$.

For the left hand side, $q_{\mathbf{I}} \lhd \varepsilon \succeq \mathsf{proc}_{\mathbf{I}} \lhd \overline{\mathsf{sub}(G)_{\mathbf{I}}}$ implies $q = \mathsf{proc}$ and $\varepsilon \preceq \overline{\mathsf{sub}(G)_{\mathbf{I}}} = \bar{p}_{\mathbf{I}}$. for the right hand side if $G$ has a $\mu$-transition then of course $\mathsf{sub}(\mu) = \mathsf{sub}(G) = p$, so $\overline{\mathsf{sub}(\mu)_{\mathbf{I}}} = \bar{p}_{\mathbf{I}}$ that we just showed to be weaker or equal to $\varepsilon$. □

## 8.2 Determinism

A *deterministic* process, also called *confluent* in the literature for reasons that will soon become clear, is in essence a process that does not perform choices, or one that has no contradictory strategies, to use the terminology introduced in Section 7.

How can we detect the presence of choice? A disjunction in the process type is neither sufficient (a deterministic process that always provides a resource $\alpha$ also always provides $\alpha \vee \beta$) nor necessary (non determinism may occur on parts of process behaviour that is irrelevant to the other properties being studied). Furthermore, existence of contradictory selection strategies is not a behavioural property as it involves inspecting the process.

A solution is to use *confluence* as a characterisation of determinism. A process $P$ is confluent if, for any pair of *distinct* transitions $P \xrightarrow{\mu_1} P_1$ and $P \xrightarrow{\mu_2} P_2$, there is a process $Q$ such that both $P_1 \xrightarrow{\hat{\mu}_2} Q$ and $P_2 \xrightarrow{\hat{\mu}_1} Q$ (where $\hat{\mu}_i$ is $\mu_i$ with possibly fewer bound names), up to renaming on bound names. “Distinct” in that sentence means that $\mu_1$ and $\mu_2$ aren't the same transition with possibly different parameters, i.e. the corresponding event pairs $(l_i|l_o)$ are different:

**Definition 8.2.1 (Determinism)** *The* $\mathsf{good}_{\mathbf{D}}$ *predicate is obtained following Definition 7.7.1 with* $\varphi(\varepsilon, (\Gamma; P))$ *being true if*

- $(\Gamma; P) \xrightarrow{\mu}$ *with* $\mu \neq \tau$ *implies* $\overline{\mathsf{sub}(\mu)_{\mathbf{D}}} \succeq \varepsilon$*, and*
- *for any pair of transitions* $(\Gamma; P) \xrightarrow{\mu_i} (\Gamma_i; P_i)$ *(*$i \in \{1, 2\}$*) such that* $\pi_1 \neq \pi_2$ *are the corresponding steps, there is a pair of transitions* $(\Gamma_i; P_i) \xrightarrow{\hat{\mu}_{\bar{\imath}}} (\Gamma'; P')$ *(where* $\bar{\imath} = 3 - i$*) such that the step corresponding to* $\hat{\mu}_i$ *is* $\pi_i$*.*

We say $(\Gamma; P)$ is *locally deterministic* if the second condition holds.

To instantiate the type system for determinism, we detect the two sources of choice, namely multiple communication partners, and sums.

Similarly to activeness, determinism of a branching is guaranteed by a process type having no concurrent environment $p_i$ (Definition 6.3.4) as a third-party process can't attempt selecting more than one branch of the sum, thereby introducing a race condition and non-determinism.

**Definition 8.2.2 (Determinism Elementary Rules)** *The elementary determinism guard rule is obtained following Definition 7.7.2 with*

$$\varphi_{\mathbf{D}}(\sigma, G, m, m') \overset{\text{def}}{=} \begin{cases} \bot & \textit{if } \star \in \{m, m'\} \textit{ and } \omega \notin \{m, m'\} \\ \overline{\mathsf{sub}(G)_{\mathbf{D}}} & \textit{otherwise} \end{cases}$$

*The elementary determinism sum rule is obtained with*

$$\varphi_{\mathbf{D}}(\{p_i\}_i, \Xi) \overset{\text{def}}{=} \begin{cases} \bot & \textit{if } \Xi \textit{ has concurrent environment } p_i \\ \top & \textit{otherwise} \end{cases}$$

See how the rule actually declares a process like $P = t.T + f.F$ to be deterministic (if it has no concurrent environment), although one may think at first sight that it is not confluent (as $P \xrightarrow{t} T$ and $P \xrightarrow{f} F$ can't be joined back). The trick is given by the projection relation. The reader will easily check that $P$'s type is

$$\Gamma = \big(t : (), f : (); \mathsf{proc}_{\mathbf{D}} \triangleleft (\bar{t}_{\mathbf{D}} \wedge \bar{f}_{\mathbf{D}}) \wedge \big((t \wedge \Gamma_T) \vee (f \wedge \Gamma_F)\big) \blacktriangleleft \bar{t} \vee \bar{f}\big)$$

That type has two projections $\Gamma_t = \big(t : (), f : (); \mathsf{proc}_{\mathbf{D}} \triangleleft (\bar{t}_{\mathbf{D}} \wedge \bar{f}_{\mathbf{D}}) \blacktriangleleft \bar{t}\big)$ and $\Gamma_f = \big(t : (), f : (); \mathsf{proc}_{\mathbf{D}} \triangleleft (\bar{t}_{\mathbf{D}} \wedge \bar{f}_{\mathbf{D}}) \blacktriangleleft \bar{f}\big)$, which respectively force the process to select the $t$-branch or the $f$-branch, i.e. both $(\Gamma_t; P)$ and $(\Gamma_f; P)$ are locally deterministic following Definition 8.2.1.

**Lemma 8.2.3 (Determinism Soundness)** *The determinism instance satisfies the conditions given in Definition 4.4.1*

*Proof* Again, requirements 1 and 3 hold by construction. The first part of Definition 8.2.1 is shown precisely like with isolation. For the second part, let two transitions $(\Gamma; P) \xrightarrow{\mu} \Gamma_i P_i$ with corresponding (and distinct) liveness strategy steps $\pi_i$, and fix the statement $q_{\mathbf{D}} \triangleleft \varepsilon$ we need to prove, with $\varepsilon \not\cong \bot$.

First assume the $\pi_i$ do not share an event, they correspond to two disjoint communicating guard pairs. For determinism to hold they must not be contradictory (Definition 7.3.3), i.e. not involve two branches of the same sum. Assume instead $\pi_1$ and $\pi_2$ respectively contain events $\mathfrak{l}_1$ and $\mathfrak{l}_2$ that are contradictory sum guards, with subjects $p_1$ and $p_2$. However, as $\varepsilon \not\cong \bot$, by the sum rule in Definition 8.2.2 the sum must not have concurrent environment multiplicities, so one of $p_1$ and $p_2$ (let's say $p_1$) is not observable in the sum, which means the sum can't be composed with a process containing a guard with subject $\bar{p}_1$, so $\mathfrak{l}_1$'s communication partner must be $\bullet$, i.e. $\pi_1$ is a labelled transition with subject $p_1$, but once more that contradicts $p_1$ being non-observable (remember that $\Gamma$ is elementary, so it can't have $\bar{p}_1 \vee \bar{p}_2$ in its environment component, as discussed after Definition 8.2.1).

Assume $\pi_1$ and $\pi_2$ share an event $\mathfrak{l}$, corresponding to some guard $G^{\mathfrak{l}}$. Let $\mathfrak{l}_1$ and $\mathfrak{l}_2$ be $\mathfrak{l}$'s two communication partners. As $\pi_1 \neq \pi_2$, $\mathfrak{l}_1 \neq \mathfrak{l}_2$. By the (COM) rule of the labelled transition system, $\mathsf{sub}_P(\mathfrak{l}_1) = \mathsf{sub}_P(\mathfrak{l}_2) = \overline{\mathsf{sub}_P(\mathfrak{l})}$. We conclude that the subject port of $\mathfrak{l}_1$ and $\mathfrak{l}_2$ has plain multiplicity. As $\varepsilon \ncong \bot$, by Definition 8.2.2 the elementary rule requires either $\omega \in \{m, m'\}$ or $\star \notin \{m, m'\}$. In the first case, as $\mathfrak{l}_i$ have multiplicity $\star$, $\mathfrak{l}$ must have multiplicity $\omega$, i.e. be replicated, so that it is still available to communicate with the other $\mathfrak{l}_i$, as required. If $\omega \notin \{m, m'\}$ then $\star \notin \{m, m'\}$, which contradicts $\mathfrak{l}_i$ having multiplicity $\star$. □

## 8.3 Reachability

We study in this section a property which is in some sense the negation of activeness.

We say a port $p$ is *inactive* or *unreachable* (written $p_{\mathbf{N}}$) in a process if it never appears in subject position. If the $\bar{a}$ output is unreachable then no continuation of an $a$-input will ever be reached, which is useful in two ways.

Suppose a program calls a particular error handling routine whenever something goes wrong, and a reachability type system proves that routine never actually gets called. Then we just proved that particular error condition never happens. Another application is *dead code elimination.* When building a program by assembling various components and libraries there may be parts that are never used. If any one component is unreachable (more specifically, its complement is unreachable), then it may be safely dropped from the program while preserving useful functionality.

A dependency analysis permits dropping "islands" of inter-dependant components: Suppose module $A$ calls module $B$, i.e. an $a$-guarded process contains a $\bar{b}$-output. A naive dead code elimination would then fail to detect that $B$ is unused, because of the $\bar{b}$-output (applying dead code elimination repeatedly would solve this particular problem, unless there are circularities. See the discussion on circularities later on). On the other hand, the reachability elementary rule given below then produces the statement

$$b_{\mathbf{N}} \lhd (\bar{a}_{\mathbf{N}} \vee \bar{b}_{\mathbf{N}})$$

which means (reading from right to left) "if the environment doesn't invoke $A$ or $B$ then $B$ can be dropped."

The semantic definition is straightforward:

**Definition 8.3.1 (Non-Reachability)** *The* $\mathsf{good}_{\mathbf{N}}$ *safety predicate is defined as follows:*

$\mathsf{good}_{\mathbf{N}}(p \lhd \varepsilon, (\Gamma; P))$ *is true if either* $\varepsilon \cong \bot$ *or* $(\Gamma; P) \xrightarrow{\mu}$ *implies* $\mathsf{sub}(\mu) \neq p$.

A process at top-level is reachable, or "not unreachable":

**Definition 8.3.2 (Elementary Non-Reachability Rule)**

$$\mathsf{prop}_{\mathbf{N}}(G, \sigma, m, m') \overset{\text{def}}{=} \mathsf{sub}(G)_{\mathbf{N}} \lhd \bot$$

Finally, in order to consume a guard, its complement must be reachable, or, more precisely, the continuation is *unreachable* if the guard's complement is. Since $\mathbf{N}$ is a universal property, this is not used by the semantics but only by the type system's (E-Pre) rule.

**Definition 8.3.3 (Reachability Transition Dependency)**
*The reachability dependencies of a guard $G$ are given by*

$$\mathsf{dep}_{\mathbf{N}}(G) \overset{\text{def}}{=} \overline{\mathsf{sub}(G)}_{\mathbf{N}}$$

**Lemma 8.3.4 (Reachability Soundness)** *The reachability instance satisfies the conditions given in Definition 4.4.1*

*Proof* Points 1 and 3 of the definition hold by construction.

Let $\Gamma \vdash_{\mathbf{N},\mathbf{ok}} P$. Remember (Convention 4.2.2) that, for any port $p$ for which $\Gamma$ contains no statement $p_{\mathbf{N}} \lhd \varepsilon$, $\Gamma$ is considered to contain an implicit statement $p_{\mathbf{N}} \lhd \top$. As the reachability elementary rule only produces statements of the form $p_{\mathbf{N}} \lhd \bot$, the premise $q_k \lhd \varepsilon \succeq \Xi_{\mathrm{L}}$ of point 2 in Definition 4.4.1 really means $q_{\mathbf{N}}$ is *not* covered by any elementary rule, i.e. $P$ contains no guard at top-level with subject $q$ and therefore $P$ can't have a transition with subject $q$. □

## 8.4 Termination

Although the concept of "termination" seems intuitively simple (a process eventually ceases all activity), there are difficulties in defining the statement "$P$ (eventually) terminates" precisely. Possible definitions (from the strongest to the weakest, and using "reduction" in the sense $\tau$-reduction) include

- "There is a number $n$ such that all reduction sequences from $P$ have length at most $n$"
- "All reduction sequences from $P$ have finite length"
- "There is a number $n$ such that, for all $P \Rightarrow Q$ there is a sequence $Q \Rightarrow A$ of length at most $n$ such that $A$ has no reductions"
- "For all $P \Rightarrow Q$ there is a sequence $Q \Rightarrow A$ such that $A$ has no reductions"

Reading the above definitions it may seem that termination is a liveness property, which would suggest the definition "$P$ eventually reaches a state with no further reduction.", where eventually is defined in Definition 5.2.2, but it turns out this is difficult to implement as a dependency analysis. For instance the process $!\,a$ terminates *unless* composed with a process with an infinite supply of $\bar{a}$-outputs, and $\bar{a}.\Omega$ terminates *unless* the $\bar{a}$-prefix is consumed. These examples, and in particular the second, show that termination is not an existential property that becomes available by putting the process to a certain state, but rather by *avoiding* a certain state, which is what the universal properties are good at.

We therefore generalise the non-reachability type system and semantics given above, in two ways

We add a port called $\tau$, which is the subject of $\tau$-transition (i.e. we set $\mathsf{sub}(\tau) \overset{\text{def}}{=} \tau$ where the first $\tau$ is a transition label $\mu = \tau$ and the second is a port $p = \tau$.

Secondly, in addition to $\mathbf{N}$ (never used in subject position) we define the $\varpi$ property, where $p_\varpi$ means $p$ is used at most a finite number of times (note that $\varpi$ looks like a slashed $\omega$).

The semantic predicate of $\mathbf{N}$ is given by Definition 8.3.1, without the $\mu \neq \tau$ condition, and with $\mathsf{sub}(\tau) \overset{\text{def}}{=} \tau$.

The semantics of $p_\varpi$ is given by studying infinitely long transition sequences and checking $p$ eventually stops appearing:

**Definition 8.4.1 (At Most Finite)** *The finiteness semantic predicate* $\mathsf{good}_\varpi$ *is such that* $\mathsf{good}_\varpi(p \lhd \varepsilon, (\Gamma; P))$ *holds if, having* $\varepsilon$*'s normal form be* $\bigvee_{i \in I} \varepsilon_i$*, for all* $i \in I$*, for all numbers* $n_q$ *there is a number* $n_p$ *such that all transition sequences*

$$(\Gamma; P) \xrightarrow{\mu_0} \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} (\Gamma'; P') \tag{8.1}$$

*containing* $n_p$ *transitions* $\mu_i$ *with subject* $\mathsf{sub}(\mu_i) = p$*, there is a port* $q$ *such that* $q_\varpi \succeq \varepsilon_i$ *and (8.1) contains at least* $n_q$ *transitions* $\mu_i$ *with subject* $\mathsf{sub}(\mu_i) = q$*.*

The numbers $n_p$ and $n_q$ are used to give the semantics of the $p_\varpi \lhd q_\varpi$ statement: if the environment provides at most a finite number of $q$-transitions then the process provides at most a finite number of $p$ transition, or, conversely (as in the definition above), in order to provide an unbounded supply of $p$-transitions the process requires an unbounded supply of $q$-transitions.

The termination resource is then the universal resource $\tau_\varpi$.

This particular application of behavioural statements is being studied by Bernhard Beschow for his Master's thesis, but working on the contraposition of dependency statement, which is more readable:

Instead of $p_\varpi \lhd q_\varpi$ ($p$ is used at most a finite number of times if $q$ is provided at most a finite number of times), one can write

$$p_\omega \rhd q_\omega \tag{8.2}$$

($p$ being usable infinitely many times *requires* $q$ being provided infinitely many times), note the triangle is inverted.

As this "contraposed" notation can be translated unequivocally with the regular one (swapping $\wedge$ and $\vee$ as well as $\top$ and $\bot$ in the dependencies, and replacing $\mathbf{N}$ and $\varpi$ by properties corresponding to their negation), the algebra and semantics of that notation can be directly inferred from the one exposed in Section 4. For instance:

$$\begin{aligned}(\alpha \rhd \varepsilon_1) \odot (\alpha \rhd \varepsilon_2) &\mapsto (\neg\alpha \lhd \neg\varepsilon_1) \odot (\neg\alpha \lhd \neg\varepsilon_2) \\ &= \neg\alpha \lhd (\neg\varepsilon_1 \wedge \neg\varepsilon_2) \\ &= \neg\alpha \lhd \neg(\varepsilon_1 \vee \neg\varepsilon_2) \\ &\mapsto \alpha \rhd (\varepsilon_1 \vee \varepsilon_2)\end{aligned}$$

so those negated universal properties behave like existential properties, while preserving safety semantics!

Regarding weakening and equivalence, $\triangleright$ is covariant on the right. For instance:

$$\begin{aligned}\alpha\triangleright(\varepsilon_1 \wedge \varepsilon_2) &\mapsto \neg\alpha\triangleleft\neg(\varepsilon_1 \wedge \varepsilon_2)\\ &= \neg\alpha\triangleleft(\neg\varepsilon_1 \vee \neg\varepsilon_2)\\ &\cong (\neg\alpha\triangleleft\neg\varepsilon_1) \wedge (\neg\alpha\triangleleft\neg\varepsilon_2)\\ &\mapsto (\alpha\triangleright\varepsilon_1) \wedge (\alpha\triangleright\varepsilon_2)\end{aligned}$$

Termination of a process corresponds to the statement $\tau_\omega\triangleright\bot$ — "potentially unlimited $\tau$-transitions requires the impossible", the contraposition of $\tau_\varpi\triangleleft\top$.

For the type system, we generalise Definition 8.3.2 to also produce $\tau_{\mathbf{N}}$-statements.

**Definition 8.4.2 (Elementary Non-Reachability Rule with $\tau$)**

$$\mathsf{prop}_{\mathbf{N}}(G,\sigma,m,m') \overset{\text{def}}{=} \mathsf{sub}(G)_{\mathbf{N}}\triangleleft\bot \wedge \tau_{\mathbf{N}}\triangleleft\overline{\mathsf{sub}(G)}_{\mathbf{N}}$$

The universal type system thus instantiated is sound, by the general soundness theorem, but not very useful: it never produces any $p_\varpi$ resource! For this we need to modify the replication operator $!\,\Gamma$ to replace any statement $p_{\mathbf{N}}\triangleleft q_{\mathbf{N}}$ by $(p_{\mathbf{N}}\triangleleft q_{\mathbf{N}}) \wedge (p_\varpi\triangleleft q_\varpi)$.

Let us see some examples before this section terminates (we omit multiplicities, channel types and environment components in these types for readability)

- The single input $a$ produces the statements $a_{\mathbf{N}}\triangleleft\bot \wedge \tau_{\mathbf{N}}\triangleleft\bar{a}_{\mathbf{N}}$, that respectively say that a transition with subject $a$ may happen, and that a $\tau$-transition may happen if this process is composed with one producing an $\bar{a}$-output.

- The replication $!\,a$ of the above process gets the type $a_{\mathbf{N}}\triangleleft\bot \wedge \tau_{\mathbf{N}}\triangleleft\bar{a}_{\mathbf{N}} \wedge a_\varpi\triangleleft\bot \wedge \tau_\varpi\triangleleft\bar{a}_\varpi$, where the last two statements respectively say that an unbounded number of transitions with subject $a$ may occur, and, should an unbounded number of $\bar{a}$-outputs be provided, an unbounded number of $\tau$-reductions may become available. Note that it is important to keep the $\mathbf{N}$-resource alongside the new $\varpi$-resources, as the next example shows:

- The process $!\,a\,|\,\bar{a}$ gets the following type:

$$\begin{aligned}(a_{\mathbf{N}}\triangleleft\bot \;\wedge\; \tau_{\mathbf{N}}\triangleleft\bar{a}_{\mathbf{N}} \;\wedge\; a_\varpi\triangleleft\bot \;\wedge\; \tau_\varpi\triangleleft\bar{a}_\varpi) \odot (\bar{a}_{\mathbf{N}}\triangleleft\bot \;\wedge\; \tau_{\mathbf{N}}\triangleleft a_{\mathbf{N}}) =\\ a_{\mathbf{N}}\triangleleft\bot \;\wedge\; \tau_{\mathbf{N}}\triangleleft\bot \;\wedge\; a_\varpi\triangleleft\bot \;\wedge\; \tau_\varpi\triangleleft\bar{a}_\varpi \;\wedge\; \bar{a}_{\mathbf{N}}\triangleleft\bot \qquad (8.3)\end{aligned}$$

  The interesting bits are $\tau_{\mathbf{N}}\triangleleft\bot$ (A $\tau$-transition may happen), obtained by reducing $\tau_{\mathbf{N}}\triangleleft\bar{a}_{\mathbf{N}}$ from the left component and $\bar{a}_{\mathbf{N}}\triangleleft\bot$ from the right one, and $\tau_\varpi\triangleleft\bar{a}_\varpi$, i.e. an infinite number of $\tau$-reduction still requires an infinite number of $\bar{a}$-outputs.

- Like with other properties, recursion is not handled at all, so $a.\bar{a}$ produces the statements $\tau_{\mathbf{N}}\triangleleft\bar{a}_{\mathbf{N}}$ and $\bar{a}_{\mathbf{N}}\triangleleft\bar{a}_{\mathbf{N}}$, which are in a sense correct ("you will get a $\tau$-transition if an $\bar{a}$-output occurs" and "you will get an $\bar{a}$-output if you provide an $\bar{a}$-output"), but blindly applying the rules reduces the second one to $\bar{a}_{\mathbf{N}} \lhd \bot$, which in turn reduces the former to $\tau_{\mathbf{N}} \lhd \bot$, i.e. "a $\tau$-transition may occur (spontaneously)", which if of course incorrect, demonstrating the incompleteness of the analysis.

Reducing $\bar{a}_{\mathbf{N}} \lhd \bar{a}_{\mathbf{N}}$ to $\bar{a}_{\mathbf{N}} \lhd \top$ (typically using delays, see Section 5.5 and other applications above) would be worse as it would make the system unsound when using replication. Specifically, the reader can verify that, using this simplification, $!\,a.\bar{a} \mid \bar{a}$ would get the same type as $!\,a \mid \bar{a}$ above, including the statement $\tau_{\varpi} \lhd \bar{a}_{\varpi}$.

A promising approach is to keeping the $\bar{a}_{\mathbf{N}} \lhd \bar{a}_{\mathbf{N}}$ statement as is, with the semantics that an $\bar{a}$-output may trigger another $\bar{a}$-output. Replication of such statements must leave a $\bar{a}_{\mathbf{N}}$ on the right hand side of $\lhd$ as is, so that $!\,(\bar{a}_{\mathbf{N}} \lhd \bar{a}_{\mathbf{N}}) = \bar{a}_{\varpi} \lhd \bar{a}_{\mathbf{N}}$ ("an infinite number of $\bar{a}$-outputs requires a finite number of $\bar{a}$-outputs"), which precisely capture the behaviour of the $!\,a.\bar{a}$-process (as it still includes the $\tau_{\varpi} \lhd \bar{a}_{\varpi}$ statement, reducing to $!\,(\tau_{\varpi} \lhd \bar{a}_{\mathbf{N}}$: a single $\bar{a}$-output may trigger infinitely many $\tau$-transitions.

- Using the above handling of self-dependencies, processes like $!\,a.s.\bar{a}$ (see the section on Partial Orders in [DS06]) produces the statements $!\,\bar{a}_{\mathbf{N}} \lhd (\bar{s}_{\mathbf{N}} \wedge \bar{a}_{\mathbf{N}}) = \bar{a}_{\mathbf{N}} \lhd (\bar{s}_{\mathbf{N}} \wedge \bar{a}_{\mathbf{N}}) \wedge \bar{a}_{\varpi} \lhd (\bar{s}_{\varpi} \wedge \bar{a}_{\mathbf{N}})$. Note that the $\bar{s}_{\mathbf{N}}$ got replaced by $\bar{s}_{\varpi}$ as it is distinct from the resource $\bar{a}_{\mathbf{N}}$ on the left hand side, but $\bar{a}_{\mathbf{N}}$ remained $\bar{a}_{\mathbf{N}}$ as in the previous example. Composing with the usual $\tau_{\varpi} \lhd \bar{a}_{\varpi}$ produces the statement $\tau_{\varpi} \lhd (\bar{s}_{\varpi} \wedge \bar{a}_{\mathbf{N}})$, i.e. the process terminates unless an infinite number of $\bar{s}$-outputs and at least one $\bar{a}$-output is provided.

**Lemma 8.4.3 (Termination Soundness)** *The termination instance satisfies the conditions given in Definition 4.4.1*

*Proof* Soundness of the $\mathsf{sub}(G)_{\mathbf{N}} \lhd \bot$ statement has already been proved in the reachability instance. Regarding soundness of $\tau_{\mathbf{N}} \lhd \overline{\mathsf{sub}(G)}_{\mathbf{N}}$, let $\tau_{\mathbf{N}} \lhd \varepsilon \succeq \Xi_{\mathrm{L}}$ with $\varepsilon \not\approx \bot$. Then assume $P \xrightarrow{\tau} P'$. By the labelled transition system there must be a port $p$ such that $P$ has two guards at top-level with subjects $p$ and $\bar{p}$. But then the elementary non-reachability rule with $\tau$ would produce the statements $p_{\mathbf{N}} \lhd \bot \wedge \tau_{\mathbf{N}} \lhd \bar{p}_{\mathbf{N}} \odot \bar{p}_{\mathbf{N}} \lhd \bot \wedge \tau_{\mathbf{N}} \lhd p_{\mathbf{N}}$, which reduces to the statement $\tau_{\mathbf{N}} \lhd \bot$, a contradiction. □

## 8.5 Deadlock-Freedom

A $\pi$-calculus process is in *deadlock* if it has a sub-process attempting to communicate, but no communication partner ever becomes available. This definition is slightly stronger than the common definition (for instance used in [KSS00]) where a process having a reduction is not considered deadlocked. However such a definition considers any process $P|\Omega$ to be deadlock-free because it is always able to perform a $\tau$-transition thanks to $\Omega$.

On the other hand our stronger definition may at first sight not be a very useful definition as a deadlock-free process is either the idle process $\mathbf{0}$ or a system with divergence, for instance for every server $!\,a(x).\bar{x}\langle v\rangle$ must be kept busy with an infinite supply of dummy clients in order to have deadlock-freedom. However dependency analysis, and more specifically dependencies of the deadlock-freedom resource, contains enough information to distinguish a "true" deadlock from one which is there by design.

Similarly to the correctness resource used when working in a purely universal setting (Section 4.5), we introduce a global deadlock-freedom universal resource $\mathsf{proc}_{\mathbf{df}}$, given by the following elementary rule:

$$\mathsf{prop}_{\mathbf{df}}(G, \sigma, m, m') = \mathsf{proc}_{\mathbf{df}} \lhd \overline{\mathsf{sub}(G)}_{\mathbf{A}} \tag{8.4}$$

Semantics is based on *liveness* (Definition 5.2.6), matching the informal definition "any guard at top-level *eventually* finds a communication partner".

**Definition 8.5.1 (Deadlock-Freedom Semantic Predicate)**
*The semantic predicate for deadlock-freedom, written* $\mathsf{good}_{\mathbf{df}}(\mathsf{proc} \lhd \varepsilon, (\Gamma; P))$, *holds if either* $\varepsilon \cong \bot$ *or, for any guard* $G^{\wr}$ *at top-level in* $P$*, there is a strategy* $f$ *such that in any infinite transition sequence of the form*

$$(\Gamma; P) = (\Gamma_0; P_0) \xrightarrow{\tilde{\mu}_0}\searrow (\Gamma'_0; P'_0) \xrightarrow{f} (\Gamma_1; P_1) \cdots$$
$$\cdots \xrightarrow{\tilde{\mu}_i}\searrow (\Gamma'_i; P'_i) \xrightarrow{f} (\Gamma_{i+1}; P_{i+1}) \cdots$$

*all* $\mu_i$ *performed by* $f$ *satisfy* $\mathsf{dep}_{\mathcal{K}}(\mu_i) \succeq \varepsilon$ *and (at least) one of the transitions corresponds to a liveness strategy step* $\pi$ *containing* $\wr$.

Note that this definition uses both strategy functions (Definition 5.2.4) and liveness strategy steps, and more specifically the annotated labelled transition system (Definition 7.4.3).

For instance a replicated input $\boldsymbol{!}\, a$ produces the statement $\mathsf{proc}_{\mathbf{df}} \lhd \bar{a}_{\mathbf{A}}$, which can be read as "a term in the process is waiting for an $\bar{a}$-output". The strategy $f$ proving this is simply doing an $a$-input which matches the $\mathsf{dep}_{\mathcal{K}}(\mu_i) \succeq \varepsilon$ requirement (both sides are precisely $\bar{a}_{\mathbf{A}}$), and has a liveness strategy step $(\wr|\bullet)$.

Another example is $t.a^i \mid \bar{a}^o$ whose type is (omitting irrelevant bits) $a_{\mathbf{A}} \lhd \bar{t}_{\mathbf{A}} \odot \mathsf{proc}_{\mathbf{df}} \lhd a_{\mathbf{A}}$ that reduces to $\mathsf{proc}_{\mathbf{df}} \lhd \bar{t}_{\mathbf{A}}$. Although both $a$ is *a priori* deadlocked as it has no communication partner available, providing a $\bar{t}$-output lets $a$ and $\bar{a}$ communicating, reducing the process to $\mathbf{0}$ that is vacuously deadlock free. The strategy proving that statement first consumes the $t$-prefix (permitted thanks to the $\bar{t}_{\mathbf{A}}$ dependency) then does a $\tau$-transition corresponding to the step $(i|o)$.

All that we have seen so far could be obtained by verifying if the complements of free names are active, but the current system also detects deadlocks involving bound (or non-observable) names, the simplest example being

$$(\boldsymbol{\nu} t)\, t^l \tag{8.5}$$

that is typed as $(\boldsymbol{\nu} t)\, \mathsf{proc}_{\mathbf{df}} \lhd \bar{t}_{\mathbf{A}} = \mathsf{proc}_{\mathbf{df}} \lhd \bot$, i.e. the process is found *not* deadlock-free, no matter what resources are provided. This matches the semantic definition as well, as there is an $l$-labelled guard at top-level but the process doesn't provide any transition whose corresponding step contains $l$ (indeed the process has no transitions whatsoever), so any statement $\mathsf{proc}_{\mathbf{df}} \lhd \varepsilon$ with $\varepsilon \ncong \bot$ would be incorrect. This example demonstrates that deadlock-freedom is *not* a behavioural property because it is not preserved by bisimulation, as process (8.5) is strongly bisimilar to the idle process $\mathbf{0}$, which is deadlock-free. It may also be of interest to use this property as a channel-level property (refer to Section 7.7) Soundness follows from the existential type system (and therefore activeness) being sound:

**Proposition 8.5.2 (Deadlock-Freedom Soundness)** *For any* $\mathcal{K}$ *including* $\{\mathbf{A}, \mathbf{R}, \mathbf{df}\}$, $\vdash_{\mathcal{K}}$ *is sound.*

# Chapter 9

# Further Reading

In this section I'll present some related research, together with, when applicable, an encoding of their notation into mine, to help comparison.

## 9.1 Activeness

### 9.1.1 Sangiorgi: The Name Discipline of Uniform Receptiveness

This [San99] is one of the first papers to address the property of activeness (which they call "receptiveness"). It works on asynchronous monadic $\pi$-calculus with sums and matching (which we don't handle). A *linear receptive* name corresponds, in my terminology, to bi-linear names that are input active, like $a$ in $a^1_{\mathbf{A}}\bar{a}^1$, and an $\omega$*-receptive* name is the same, but with $\omega$ multiplicity on input and plain multiplicity on output, like $a_{\mathbf{A}}{}^{\omega} \wedge \bar{a}^\star$.

Their $(\Gamma;\Delta)$ process types can then be translated into my process types by having a name $a$'s local multiplicities be $\bar{a}^{\Gamma(a)}a^{\Delta(a)}$ for the linear type system (with $A(a) = 1$ if $a \in A$ and 0 otherwise), and the complement multiplicities $\bar{a}^{1-\Gamma(a)}a^{1-\Delta(a)}$ on the remote side. For the $\omega$-receptiveness type system, we have, for each $a$, $\bar{a}^{\star\Gamma(a)}a^{\omega\Delta(a)}$ on the local side, and $\bar{a}^\star a^{\omega(1-\Delta(a))}$ on the remote one. Sangiorgi's *plain names* correspond to $a^\star\bar{a}^\star$, both locally and remotely (names plain on both ports, and without activeness).

Note however that his type system is typing strong activeness, so that it does not require dependency analysis, but also is not subsumed by mine. If however we weaken his soundness theorem to allow a weak input transition when using a receptive name, then my semantic definition matches his, and typability of my type system strictly implies his.

He also provides definitions for labelled bisimilarity and barbed equivalence that respect the concept of receptiveness. Generalising those definitions, in particular 5.3, the one for labelled bisimilarity, would however require some work, because if receptive names are allowed to carry receptive names, then the $x \triangleright v$ sub-process is not complete.

### 9.1.2 Pierce, Sangiorgi: Typing and Subtyping for Mobile Processes

This paper [PS93] studies input and output capabilities (in my terminology, types such as $\top$, $a^\star$, $\overline{a}^\star$, and $a^\star\overline{a}^\star$), and establishes a *subtyping* relation, which permits typing $\overline{a}\langle x\rangle$ while having $x$'s type different from $a$'s parameter type (using the subtyping relation covariantly or contravariantly depending on which capabilities of $x$ are used by $a$'s receiver).

Their types $(\tilde{S})^I$ with $I \in \{\mathbf{r}, \mathbf{w}, \mathbf{b}\}$ are easily encoded into my notation, as follows:

$$[\![\, a : (\tilde{S})^I \,]\!] \overset{\text{def}}{=} \left(a : ([\![\, \tilde{S} \,]\!]); a^{\star I_{\mathbf{r}}}\overline{a}^{\star I_{\mathbf{w}}} \blacktriangleleft a^{\star \bar{I}_{\mathbf{r}}}\overline{a}^{\star \bar{I}_{\mathbf{w}}}\right)$$

where $\star I_c$ is $\star$ if $I \leq c$, 0 otherwise, where $\star\bar{I}_c$ is the same but using $c \leq I$, and $[\![\, S_1, \ldots, S_n \,]\!]$ is an abbreviation of $[\![\, 1 : S_1 \,]\!], \ldots, [\![\, n : S_n \,]\!]$.

Their types are thus more specific (all names are plain and none can be declared active) but, with equivalent types, their type system accepts more processes than mine, thanks to subtyping.

### 9.1.3 Kobayashi, Pierce, Turner: Linearity and the $\pi$-calculus

That paper [KPT99] is a specialisation of my system in that they only have inert (multiplicity zero), linear (only one port is used, and linearly), bi-linear (both ports are linear) and plain names (which they call $\omega$), and no behavioural property. They also introduce $(\omega; \star)$ channels in section 7.3 (and call them $*$). Like in Section 9.1.2, we can encode their types as follows:

$$[\![\, a : p^m[\tilde{T}] \,]\!] \overset{\text{def}}{=} \left(a : ([\![\, \tilde{T} \,]\!]); a^{[\![\, m \,]\!] p_i}\overline{a}^{[\![\, m \,]\!] p_o} \blacktriangleleft a^{[\![\, m \,]\!] \bar{p}_i}\overline{a}^{[\![\, m \,]\!] \bar{p}_o}\right)$$

where $mp_c$ is $m$ if $c \in p$, 0 otherwise, $[\![\, 1 \,]\!] \overset{\text{def}}{=} 1$, and $[\![\, \omega \,]\!] = \star$. $[\![\, T_1, \ldots, T_n \,]\!]$ is an abbreviation of $[\![\, 1 : T_1 \,]\!], \ldots, [\![\, n : T_n \,]\!]$.

They provide definitions for barbed bisimilarity, and show some confluence results for linear channels.

### 9.1.4 Amadio et al.: The Receptive Distributed $\pi$-calculus

As the title suggests, this paper [ABL03] is on a distributed setting, where they have the additional issue that, for a communication to succeed, its two ends must be at the same site (which requires extra care when checking for deadlocks). They also have matching, on a special set of names called *keys*.

So, the setting is more complex, with the trade off that their types are very simple — all names are (in my terminology) active non-uniform $\omega$ input and plain output and, just like [San99], they guarantee *strong* activeness, where no internal action is tolerated between creation of a new name and it being ready to use). More importantly, as a consequence of having I/O alternation and only input activeness, they are only concerned about messaged being *received* — no reply is guaranteed.

Their work is mainly interesting in the distributed setting — restricting it to a local setting would reduce to the essentially syntactic check that all outputs have at least one corresponding unguarded input.

Also note that they concentrate on *non-uniform* activeness based on recursion (like $a$ in $\mu X.a(x).(\overline{x}\langle t\rangle \mid a(y).(\overline{x}\langle t'\rangle \mid X))$ where $\mu X.P$ stands for a recursive process), which can't be characterised in my type system without modification, as the closest we have is *uniform* activeness obtained through replication.

### 9.1.5 Acciai, Boreale: Responsiveness in process calculi

This paper [AB08a] addresses concerns very close to activeness (Section 6), through two distinct type systems. Note that what they call "responsiveness" mostly corresponds to what I call "activeness". Again, their setting is simpler than mine, in that it works on synchronous $\pi$, I/O alternating, doesn't consider combinations of active and non-active names, and does not support choice or conditional properties, as it uses numerical levels to track dependencies. On the other hand, they present, with their system $\vdash_1$, an extension for recursive processes which is more powerful than my type system, in that it permits handling unbounded recursion such as a function computing the factorial of its parameter: $\boldsymbol{!}\, f(n,r).\, \text{if}(n=0)\, \overline{r}\langle 1\rangle \text{ else}\, (\boldsymbol{\nu} r')\, (\overline{f}\langle n-1,r'\rangle \mid r'(m).\overline{r}\langle n*m\rangle)$. My type systems reject such a process, because the recursive call would create a dependency $f_{\mathbf{R}} \triangleleft f_{\mathbf{R}}$.

I conjecture that their analysis, based on the well-foundedness of parameter domains, could be adapted to my behavioural statements by having a $a_{\mathbf{R}} \triangleleft b_{\mathbf{R}}$ dependency using *delays* (Section 5.5) typing $\boldsymbol{!}\, a(\tilde{y}).\overline{b}\langle\tilde{x}\rangle$ with $a_{\mathbf{R}} \triangleleft {b_{\mathbf{R}}}^{d}$ where $d > 0$ only if $\tilde{x}$ is "lighter" than $\tilde{y}$. A circular dependency chain containing only such dependencies reduces to $\top$ rather than $\bot$. In the factorial example, $\langle n-1, r'\rangle$ being "lighter" than $\langle n, r\rangle$ (because $n-1 < n$), the self-dependency becomes $f_{\mathbf{R}} \triangleleft {f_{\mathbf{R}}}^{1}$ and cancels out into $f_{\mathbf{R}} \triangleleft \top$.

**Types** A channel type can be *responsive*, *$\omega$-receptive* or *+-responsive*. For the last case they use a concept mostly equivalent to multiplicities, which they call "capabilities". Their channel types can then be encoded into mine as follows:

- Inert type: $[\![\, a : I \,]\!] = \big(a : ();\ \blacktriangleleft a^0 \wedge \bar{a}^0\big)$
- Responsive name: $[\![\, a : T^{[\rho,k]} \,]\!] = \big(a : ([\![\, 1 : T \,]\!]); a_{\mathbf{A}} \wedge \bar{a}_{\mathbf{A}} \blacktriangleleft a^0 \wedge \bar{a}^0\big)$
- Responsive parameter: $[\![\, 1 : T^{[\rho,k]} \,]\!] = (1 : ([\![\, 1 : T \,]\!]); \bar{a}_{\mathbf{A}} \blacktriangleleft a_{\mathbf{A}})$
- $\omega$-receptive name: $[\![\, a : T^{[\omega,k]} \,]\!] = \big(a : ([\![\, 1 : T \,]\!]); a^{\omega}_{\mathbf{A}} \wedge \bar{a}^{\star} \blacktriangleleft a^0\big)$
- $\omega$-receptive parameter: $[\![\, 1 : T^{[\omega,k]} \,]\!] = (1 : ([\![\, 1 : T \,]\!]); \bar{a}^{\star} \blacktriangleleft \wedge)\, {a_{\mathbf{A}}}^{\omega}$
- +-responsive names are encoded similarly, using the following correspondence: on inputs, capabilities $n$, $s$, $m$ and $p$ correspond respectively to total multiplicities 0, 1, $\star$ and $\omega$, and on outputs, $n$, $s$, $m$ and $p$ correspond respectively to total multiplicities 0, $\star$, $\star$ and $\omega$.

We have no way to prevent a name to be sent around (in object position), so their $\bot$ type can't be encoded. Encoding it like $I$ is a good approximation, however. Also, their levels $k$ are ignored by this encoding, because they are implicitly contained in the behavioural statement which is inferred by the type system. Those levels basically put an upper bound on the length of substitution chains $(\{^{\beta}/_{\alpha}\}\{^{\gamma}/_{\beta}\}\cdots)$ that can be done in activeness dependencies before reaching the $\top$-dependency. The above encoding is not completely accurate but corresponds to what their type system enforces.

**Semantics** As far as terminology is concerned, their "responsiveness" property mostly corresponds to my "activeness" property, on processes in which responsiveness (in my terminology) holds on all names. It is not strictly equivalent because we work with a labelled transition system and define activeness and responsiveness in terms of interactions with the environment, while they work in a reduction setting, and define responsiveness in terms of internal actions. The correspondence can be made by comparing my activeness on a port $p \in \{a, \bar{a}\}$ in a process $P$ to their responsiveness on channel $a$ in a process like $P \,|\, Q$ where $Q$ is a process interacting on $\bar{p}$ (such as $\bar{a}\langle b\rangle$ or $a(x).Q'$, depending on $p$).

Note that their semantic definition is also weaker as it accepts as responsive channel $a$ in "unbalanced" processes like $(a \,|\, \bar{a}) \,|\, a$ or $(a \,|\, \bar{a}) \,|\, \bar{a}$, where the rightmost $a$ or $\bar{a}$ can be seen as the "testing" process $Q$, but may not succeed. Also they require more than fairness on the scheduler as they would consider $s$ responsive in process $P_6 \,|\, s$ (where $P_6$ is given by equation (5.4)). However it seems that strengthening their semantic definition to reject such cases would preserve soundness of their type system.

It should be noted also that they require *all* names to be "responsive" (or $\omega$-receptive, which is essentially the same but with another multiplicity) — they don't consider processes where both "plain" and "responsive" names are involved.

**Power** The base form of both their type systems, described in their sections 3 and 6 are strictly subsumed by mine.

Similarly to what was presented in this paper, their first type system uses a behavioural statement is used to check strong linear activeness or strong $\omega$-activeness on input ports, and activeness for linear output ports. For a process like $\bar{b} \,|\, b.\bar{a}$, a dependency $a \to b$ indicates the order in which linear channels are consumed. it uses *levels* to check delegation, in a way that corresponds more or less to my *responsiveness* dependency chains, e.g. $!\, a(x).\bar{b}\langle x\rangle$ requires $b$'s level to be smaller than $a$'s.

Their first system rejects a number of processes accepted by my type system, such as "half-linear names" like $t$ in $(\boldsymbol{\nu} t)\ (\bar{t} \,|\, t.P \,|\, t.Q)$, as well as processes such as $(\boldsymbol{\nu} a)\ (a(x).(\bar{x} \,|\, b(y).\bar{y}) | \bar{a}\langle t\rangle)$ because the input on $b$ is not immediately available. It is however weakly bisimilar to $b(y).\bar{y}$, which is typable.

On the other hand the extension for handling recursive functions goes beyond what my type system is capable of, as already said.

The second type system allows guarded inputs, the "half-linear names" already mentioned and replicated outputs, but rejects some recursive functions such as the "factorial" one given previously. It is also strictly subsumed by mine because for instance they do not allow guarded free replicated inputs.

We would like to point out that this paper answers the question they rise at the end of Section 6.2, concerning the generalisation of dependency graphs when inputs may be nested. They give an example of process that would require such a generalisation: $b(x).\bar{a}\langle x\rangle \,|\, c(x).a(y).\bar{x}\langle y\rangle \,|\, \bar{c}\langle b\rangle$, where all names are assumed responsive (in their terminology, or "bi-linear active" in mine). That process should be ruled out because it reduces to $b(x).\overline{a}\langle x\rangle \mid a(y).\overline{b}\langle y\rangle$, where $a$ and $b$ are now clearly deadlocked. Using dependency graphs on responsiveness (in addition to activeness) rules out the first process, because it contains the cycles $b_{\mathbf{R}} \leq \bar{c}_{\mathbf{R}} < a_{\mathbf{R}} < b_{\mathbf{R}}$ and $c_{\mathbf{R}} \lhd \bar{a}_{\mathbf{R}} \lhd \bar{b}_{\mathbf{R}} \lhd c_{\mathbf{R}}$.

In conclusion, generalising their analysis of recursion on well-founded domains on my type system would give a type system that is strictly more powerful than both their systems, so that it is no longer necessary to have two separate systems with different typing strategies.

### 9.1.6 Kobayashi: TyPiCal

This [Kob08] is an implementation of a lock-freedom type system [Kob02a]. Although it also performs termination and information flow analysis we are particularly interested in its lock-freedom analysis.

**Terminology** We first introduce a few concepts used by TyPiCal.

**Definition 9.1.1 (Deadlock)** *An input or output prefix in a process $P$ is* deadlocked *if it is top-level and $P$ can't be reduced.*

*An input or output prefix in a process $P$ is* deadlock-free *if no reduction of $P$ leads to that prefix being deadlocked.*

For example, if $\nexists Q : P \rightarrow Q$ then all top-level actions in $P$ are deadlocked. In $!\,a(x).P \mid Q$, all $a$-outputs are deadlock-free. In $a.\bar{b} \mid b.\bar{a}$, both $a$ and $b$ are deadlocked. In $P = ?.a \mid \bar{a}$, $a$ is deadlock-free, but $\bar{a}$ isn't ($P \rightarrow\equiv \bot.a \mid \bar{a}$ in which $\bar{a}$ is deadlocked, although $P \rightarrow\sim a \mid \bar{a}$ in which $\bar{a}$ is deadlock-free).

Deadlock-freedom is not a very interesting property on its own, because for instance $P|\Omega$ is deadlock-free as it can always be reduced.

One way would be to require all processes to terminate, but a more general approach is introduce to the following (strictly stronger) property:

**Definition 9.1.2 (Livelock-freedom)** *An action of a process $P$ on a port $p$ is* livelock-free *if it reaching top-level implies it can be consumed.*

For example, a request to a server is livelock-free is and only if it is guaranteed to be eventually received. In $!\,a(x).\bar{x} \mid \overline{a}\langle b\rangle \mid b$, the input at $b$ is livelock-free, and in $P = \,!\,a(x).\overline{b}\langle x\rangle \mid !\,b(x).\overline{a}\langle x\rangle \mid \overline{a}\langle s\rangle \mid s$, the $s$-input is deadlock-free but not livelock-free.

This property is related to activeness in that (although either definition need to be adapted as we work in a labelled setting and TyPiCal in a reduction setting) $p$ is livelock-free if and only if the complement port $\bar{p}$ is active.

*Channel usages* are a generalisation of multiplicities, and tell for a particular channel how many times the input and output ports are used, and in what order.

**Definition 9.1.3 (Channel Usages)** *The* usage *of a channel is an expression given by the following grammar:*

$$U ::= \mathbf{0} \;\big|\; \rho \;\big|\; u.U \;\big|\; (U|U) \;\big|\; U\&U \;\big|\; \mu\rho.U$$
$$u ::= \,! \;\big|\; ?$$

Usage $!.U$ does an output and then $U$; Usage $?.U$ does an input and then $U$. $(U_1|U_2)$ uses according to $U_1$ and $U_2$ in parallel. $U_1\&U_2$ uses according to either $U_1$ or $U_2$ but not both. We write $\mathrm{chan}_U(\tilde{\sigma})$ for a channel of usage $U$ and parameters $\tilde{\sigma}$. When the context is clear, we may write just the usage for a parameter-less channel.

For example, $a.b \,|\, \bar{b}.\bar{c}$ uses $a$ according to $?$, $b$ according to $?|!$ and $c$ according to $!$. In $\boldsymbol{!}\, a(x).\overline{x}\langle 1\rangle$, $a$ has usage $*?|!$ (with $*? \overset{\mathrm{def}}{=} \mu\rho.(?.\rho)$), and thus $\mathrm{chan}_{*?|!}(!), b :!$ as a channel type (the parameter usages give the behaviour of the channel's *input* side, and here the $a$-input *outputs* on $x$). As a last example, say $a \neq t$ has usage $U_1$ in $P$ and $U_2$ in $Q$. It then has usage $U_1\&U_2$ in $(\boldsymbol{\nu} t)\,(\bar{t} \,|\, t.P \,|\, t.Q)$.

Obligation and Capability levels generalise the levels used in [AB08a]:

**Definition 9.1.4 (Obligation and Capability Levels)** *An* obligation level *for an (input or output) primitive is a number (or $\infty$) telling when it will be ready to fire (i.e. at top-level), while a* capability level *tells, if that primitive is at top-level, when it will actually be consumed.*

These levels are included into usages with the syntax $u \;::=\; !^{t_O}_{t_C} \;\;|\;\; ?^{t_O}_{t_C}$.

For example, consider the process $a.b \,|\, \bar{b}.\bar{c}$. The input $a$ is at top-level and thus has obligation level 0: Assuming it gets consumed at time $t$, $b$ will be ready to fire at time $t+1$. The output $\bar{b}$ is immediately ready, but will actually get consumed at time $t+1$. $b$ has capability 0 because no matter when it is brought to top-level, $\bar{b}$ will be ready to communicate with it. To sum up, we get the following: $a : (?^0_t), b : (?^{t+1}_0|!^0_{t+1}), c : (!^{t+2}_{t'})$.

In this example, the obligation level of a port is equal to the capability level of its complement. However this is not always the case in presence of non-linearity: In $\bar{a}.x \,|\, \bar{a}.y \,|\, a.z \,|\, \bar{x}$, $a$ has usage $(!^0_\infty|!^0_\infty|?^0_0)$ — both $\bar{a}$ have capability zero because neither is guaranteed to succeed. Being at top-level, all $a$ and $\bar{a}$ have obligation zero.

As expected, activeness, responsiveness, livelock-freedom, obligation and capability levels are tightly related:

- A term is active if and only if it has a finite obligation level and all complement actions have a finite capability level.
- A term is strongly active if and only if it has a zero obligation level and all complement actions have a zero capability level.
- A term is livelock-free if and only if it has a finite capability level.
- Input (resp., output) responsiveness corresponds to finiteness of all obligation (resp., capability) levels on parameter usages.

**Power** There is no subsumption relation either way between my system and the one implemented by TyPiCal.

On the one hand, the usage information is strictly more expressive than multiplicities (which can mostly be encoded in terms of usages, with the slight difference that usages can't express the *uniformity* inherent to $\omega$-multiplicity). This permits for instance TyPiCal to handle locks correctly, as well as processes like $a \mid a.\bar{s} \mid \bar{a} \mid \bar{a}$ (where $\bar{s}$ is active because $a$'s input and outputs are balanced, unlike for example $b$ in $b \mid b.\bar{s} \mid \bar{b} \mid \bar{b} \mid \bar{b})$. Multiplicities would dismiss locks as well as that port $a$ as plain names.

On the other hand, Events described in Section 5.4 permit an accurate analysis of processes such as

$$(\boldsymbol{\nu} t)\,\big(\bar{t}\,\big|\, t.(!\, z|!\, a(x).\bar{z}.\bar{x})\,\big|\, t.!\, a(y).\bar{y}\big)$$

which randomly picks a "slow" or a "fast" $a$-input. TyPical incorrectly marks the $\bar{z}$ output as unreliable (not livelock-free). Labels make $z$'s unreliability (or non-activeness, or infinite obligation level) irrelevant when checking $a$'s responsiveness.

It should be noted that neither my system nor TyPiCal recognises $a$ as input active in that process, which suggests a future research direction.

Finally, TyPiCal does not handle recursive channel types that would be required to analyse processes like $\overline{a}\langle a\rangle$ or $!\, a(x).\overline{x}\langle a\rangle$ but we believe it would be a rather simple extension, as was the case for my system.

### 9.1.7 Kobayashi: Type Systems for Concurrent Programs

This paper [Kob02b] covers most of the theoretical basis (including channel usages, capability and obligation levels) for TyPiCal, in the form of a type system being described incrementally, similarly to the present paper. The analysis given in Section 9.1.6 therefore remains mostly valid, except that [Kob02b] works on polyadic $\pi$. It also covers tail recursive functions (similarly to [AB08a]), and a number of interesting extensions such as *session types* and *termination* analysis.

Their types don't seem to describe a separation of input and output protocols in channel types.

My strategy of using explicit behavioural statements instead of obligation (and capability) levels has the advantage of describing a process as an open system, in that it describes how the process would react when composed with an arbitrary other process. For instance, if $P = a.b$, then seeing $P$ as a closed system implies that $b$ will never be available. Describing it with a behavioural statement makes explicit in the type that $b$ becomes active if $\bar{a}$ is.

### 9.1.8 Kobayashi and Sangiorgi: A Hybrid Type System for Lock-Freedom of Mobile Processes

This paper [KS08] combines (arbitrary) deadlock, termination and confluence type systems on *sub-processes* of the one being analysed (thereby permitting analysis of globally divergent processes). This work uses typed transitions reminiscent of mine, and their "robust" properties are analogous to my semantics permitting arbitrary transition sequences $\tilde{\mu}_i$. Channel usages are like those used by Kobayashi in previous works [Kob02a, Kob08], with the same expressive power and limitations. The typing rules discard those processes that rely on the environment in order to fulfil their obligation. Hence well-typed processes are lock-free without making any assumption on the environment. Advanced termination type systems such as those proposed by Deng and Sangiorgi [DS06] permit this hybrid system to deal with complex recursive functions like tree traversal.

## 9.2 Other Properties

### 9.2.1 Deng and Sangiorgi: Ensuring Termination by Typability

This paper [DS06] proposes a series of increasingly powerful type systems for characterising termination. The definition of termination (all reduction sequences are of finite length) coincides with $\tau_\varpi \triangleleft \varepsilon$ where $\varepsilon \preceq p_\varpi$ for some $p$. The first system is quite basic and worked by attaching levels to channels, and I believe it is equivalent to the one we discussed in Section 8.4. Much like level-based lock-freedom type systems, levels correspond to the length of dependency chains. Unlike dependency analysis, levels must be provided as part of the channel types.

They also provide a direct way of computing an upper bound on the number of reduction a terminating process may do, which is something my termination instance can't do as it works on universal properties. (In contrast with the existential type system, whose annotated form produces strategies that have a well-defined weight).

The paper then proceeds to typing recursive processes, much like [AB08a] does for activeness, by recognising recursive calls carrying a "lighter" parameter. Again, it seems recognising this sort of well-founded recursion could be added to the generic type system, by choosing the delay accordingly (Section 5.5).

In section 5 they introduce rules for dealing with infinite recursion that is limited by another input, as in $!\,a.b.\bar{a}$, which can effectively be described as $\tau_\varpi \triangleleft \bar{a}_{\mathbf{N}} \vee \bar{b}_\varpi \triangleleft$ (i.e. it terminates unless provided with an infinite supply of $\bar{b}$-outputs).

### 9.2.2 On Determinacy and Nondeterminacy in Concurrent Programming

Nestmann's PhD thesis [Nes96] contains a detailed description of choice and determinism, a type system detecting non-determinism, and studies the encoding of sums $P+Q$.

## 9.3 Generic Type Systems

### 9.3.1 Acciai and Boreale: Spatial and Behavioral Types in the Pi-Calculus

This Type System [AB08b] combines ideas from the Kobayashi's Generic Type System (in that types abstract the behaviour of processes) and Spatial Logic, by performing model checking with spatial formulæ on the types rather than on the processes. This results in a generic type system able to characterise liveness properties such as activeness, and supports choice, both through the process constructor $+$ and logical connective $\vee$. It is parametrised by "shallow" (without direct access to the object parts of transitions) logical formulæ, that it checks automatically using a model-checking approach. Being based on model checking, it suffers from the same limitations as the previous work, in terms of computation complexity, and difficulty of expressing conditional properties or responsiveness (by "shallowness" of the logic — note once more that what the authors call responsiveness corresponds to what we call activeness). On the other hand, restricting it to shallow logic formulæ allows working on the abstracted process, making it more efficient than a fully general model checker. Like the previous work and unlike the Generic Type System, it doesn't require proving soundness of a consistency predicate, as it is based on a fixed formula language.

### 9.3.2 Igarashi and Kobayashi: A generic type system for the Pi-calculus

This [IK01] is a framework for type-checking various safety properties such as deadlock-freedom or race-freedom. Types are *abstract processes* — a simplified form of the target process — and soundness theorems establishing that if the abstraction is well-behaved then so is the actual process. It is particularly useful for *safety* properties as subject reduction is proved once and for all, so that instances of the generic type system only need to show that if the abstract process is well-behaved, the target process is not *immediately* breaking the desired property. With liveness properties, showing the validity of a dependency analysis done on the abstract process and the correspondence between activeness in the abstract process and the actual one would likely require the same amount of work as starting from scratch.

Types use "+" in essentially the same sense as we do, and "&" corresponds precisely to $\vee$ as used in this thesis. The paper includes as examples of instantiations, simple arity-mismatch checking system, race-freedom and deadlock-freedom type systems.

The idea behind their type system and mine is rather different as well. In a word, Igarashi and Kobayashi's system constructs an abstract process that is really a very detailed type, and then custom rules work on the types (and not at the process) and return "yes" or "no". My system, in contrast, has elementary rules that look at elements of a *process* and provides building blocks of types, that are then assembled by the type system. It is still too early, however, to decide if one approach is best or if they are simply complementary.

### 9.3.3 Caires and Vieira: Spatial Logic Model Checker

This paper [Cai04] presents a model checker able to check processes for a wide range of properties, expressed by expressions written in a *spatial logic*, and is sound and complete as long as (the state spaces of) the processes are *bounded.* Using their logic, activeness of a port $p$ can be written $\nu X.(\langle p\rangle \vee \Box\Diamond X)$. Responsiveness of a port is a property that depends on the channel type, but it should be possible to give an inductive translation of channel types to modal formulæ corresponding to responsiveness on it. The selection connective $\vee$ is also present in their logic, with the same meaning. There is however no direct equivalent to my $\lhd$ connective, so conditional properties need to be encoded by modifying the activeness formulæ, which may become very complex for statements such as (6.17) that include dependencies on responsiveness. Both its strengths and limitation come from it being purely a *model checker.* On the one hand, it takes logical formulæ in *input* rather than constructing them automatically, it has a very large complexity due to exhaustively exploring the state space, and doesn't terminate when given unbounded processes (unlike a type system such as mine, that is polynomial in the size of the process, and always terminates). On the other hand it is *complete* for bounded processes, and able to recognise activeness in cases deemed unsafe by my type system, due to over-approximation.

## 9.4 Structural Analysis

### 9.4.1 Bodei, Degano et al: Control Flow Analysis for the $\pi$-calculus

Related to structural analysis (Section 7), the theory developed in this paper [BDNN98] is focused on the following problem: $P$ being a (monadic — but the theory seems straightforward to generalise to the polyadic setting) $\pi$-calculus process, what is the set of names that can be carried by a given channel, while the process evolves? As the problem is not decidable, the authors construct an over-approximation. Note that the "semantic definition" $\models_{me}$ is really a syntax-directed type system, as exposed in Section 4, while the actual semantics that relation guarantees is given by Subject Reduction (Theorem 3.10).

They avoid the problem of $\alpha$-renaming by inserting channel and binder markers into the process syntax, then referring to channels by channel markers rather than names, rather similarly to "events" $l$. They do not require distinct channels to have distinct names, however, avoiding the need for "extended names" $\mathfrak{x}$ (and it is acceptable precisely because they construct an over-approximation). The more distinct channels are used in the process annotation, the more precise the analysis will be.

This question — what is the set of names that can be carried by a given channel — is relevant to my research in two ways.

First, for a liveness property $p_k$ to be available in a process, there needs to be a guard $G$ somewhere that provides a property $q_k$ where either $q = p$ or $q$ is bound by an input prefix somewhere, and $q$ gets *instantiated* to $p$ by a communication partner of that prefix. My liveness type system handles this with channel types and parameter instantiations, which is one of its fundamental limitations. An approach based on computing what names may be instantiated to what channels might provide a higher degree of accuracy, although we'd require an *under-approximation* for liveness to hold.

Secondly, completeness of an annotated type, that is when dealing with interference, relies on knowing all communication partners of a given guard $G$. For instance in $P = \overline{a}\langle t\rangle.t.A \mid x(y).y \mid \ldots$, if some liveness resource $\gamma$ is available in $A$, proving it is available in $P$ as well requires us to find all potential communication partners of $\overline{x}\langle t\rangle$ and check they enable an output at their parameter. Finding all $x$-inputs in a process amounts to finding all names carried by other inputs, for instance if the process contains $a(y).y(z)^{l}.\ldots$, then $l$ becomes an $x$-input if and only if $a$ carries an $x$. For this part we do need an over-approximation (but we need more than just a set of names, we need to unambiguously distinguish all potential communication partners, so some form of liveness strategy seems unavoidable).

As a chief application of the type system, [BDNN98] proposes an application to *information flow* (if the type system concludes that, in $P$, no "low channel" ever carries a "high parameter", one can conclude the process will not leak secret information).

# Chapter 10

# Conclusion

In this thesis, I proposed dependency analysis as a generic way to describe and analyse the behaviour of a $\pi$-calculus process. Channel and process types are equipped for integrating arbitrary behavioural resources, as well as semantics for existential and universal properties. A generic type system, given elementary rules characterising the essence of the desired properties, constructs process types with detailed behavioural statements summarising what properties are guaranteed by the process, and — through dependency statements — what further properties it provides given some help (in the form of existential resources) by the environment.

One strong point of dependency analysis is a high expressiveness — not only it permits encoding the types of all papers we considered (except for non trivial channel usages), it allows for a more detailed specification of the protocol being used on a channel and capabilities being transmitted.

Behavioural statements in process types accurately specify the interface of the process with the environment, so that having typed $P$ and $Q$ independently, their types can be composed to obtain $P|Q$'s type (unlike most works we surveyed, that treat processes as a whole and in a reduction-based setting and thus can't directly predict the effects of such a composition without type checking the composition itself). Similarly, the reliability built into liveness semantics (Section 5.2) means the properties are preserved by composition, which is not always the case with such reduction-based settings.

Where do we stand now regarding our original goal, producing a general framework for verifying encodings?

A typical proof of equivalence of two calculi $\mathcal{C}$ and $\mathcal{C}'$ would proceed as follows:

1. Use a simple type systems for $\mathcal{C}$ verifying that processes do not mismatch channel types, essentially the universal type system (Table 4.1) letting all channels be plain ($a^\star \wedge \bar{a}^\star$) and setting $\mathcal{K} = \varnothing$, but possibly extended with primitive types such as Integers or more complex objects.

2. Write a process encoding $[\![ \cdot ]\!] : \mathcal{C} \to \mathcal{C}'$

3. Write a mapping of $\mathcal{C}$-channel types to $\mathcal{C}'$-channel types. For instance, when encoding $\pi$-calculus with Booleans to the basic $\pi$-calculus, this mapping would replace the primitive **Bool** to $\mathsf{Bool}$ as given by (6.12).

Special channels introduced by the encoding (like $u$ in Section 1.3) must typically be declared with various behavioural properties for the encoding to be fully abstract.

4. A *typability* proof must show that if a $\mathcal{C}$-process $P$ is simply typed then so is its encoding $[\![ P ]\!]$. Then the generic soundness theorems apply, and all encoded processes are guaranteed to satisfy the properties chosen in the previous step.

5. The fully abstract theorem must then show that if two $\mathcal{C}$-processes $P_1$ and $P_2$ are *barbed congruent* if and only if their encodings $[\![ P_1 ]\!]$ and $[\![ P_2 ]\!]$ are *barbed congruent* with respect to *typable* $\mathcal{C}'$-environments.

6. The same procedure must be repeated in the $\mathcal{C}' \to \mathcal{C}$ direction, to complete the equivalence proof.

Step 5 can be rather difficult in the $[\![ P_1 ]\!] \cong [\![ P_2 ]\!] \Rightarrow P_1 \cong P_2$ direction, because not all typable environments are encodings of $\mathcal{C}$-processes. One proof strategy would be, by building on the semantic definitions of the chosen behavioural properties, to show that all typable processes are bisimilar to some encoded process, because they must only use encoded channel types when interacting with the $[\![ P_i ]\!]$. This might not always be possible, if $\mathcal{C}'$ is sufficiently rich or the encoding sufficiently elaborate.

Another research direction is therefore to design a *labelled* bisimulation relation to work on typed $\mathcal{C}'$-processes, that respects the semantic properties by only considering transitions that could be triggered by well-typed processes. The transition operator $\wr$ already guarantees this for simple types (i.e. it respects channel types and multiplicities) but currently offers no such guarantees for behavioural properties, and in particular a good labelled bisimulation should respond identically to repeated (but identical) requests of encoded processes on channels that are declared deterministic in the process type.

Regarding the limitations of this generic type system, I chose to focus on choice itself, leaving out features like recursivity [AB08a, DS06] or subtyping [PS93], and complex channel usages such as locks [Kob02b, Kob08, Kob02a], which have been well explored before in a choice-less context. With some work, these could probably be integrated into the generic type system for improved accuracy. Of course, such an inclusion would in a single step benefit all instances, described in this thesis or elsewhere, which is what makes generic type systems so appealing.

Note however that integrating recursivity to work well with encoded values would be non-trivial because nothing in an encoded Integer type prevents numbers to be infinite (which may or may not be permitted in the source calculus), and yet it may be desirable in some contexts to permit unbounded numbers. For instance

$$!\,\mathsf{Geom}(zero, succ).(\overline{zero} + \overline{succ}\langle \mathsf{Geom} \rangle) \tag{10.1}$$

is a random number obeying a geometric distribution, safe for use with arithmetic operators. Moreover, an addition operator working by induction on the first parameter would be responsive even if the second parameter is infinite. In other words, a treatment of responsiveness with recursion would have to include the concept of finiteness "**B**" (like **B**ounded, as **F** was already taken by

**F**unctional, but we really mean finite) in addition to activeness and responsiveness. The following example encodes the circuit "$r = a + b$" and shows that $r$ is responsive even if $b$ is infinite, but $r$ is finite only if both $a$ and $b$ are finite.

$$\begin{aligned} &r^{\omega}_{\mathbf{A}} \;\wedge\; r_{\mathbf{R}}\triangleleft(a_{\mathbf{AB}} \wedge b_{\mathbf{AR}}) \;\wedge\; r_{\mathbf{B}}\triangleleft(a_{\mathbf{AB}} \wedge b_{\mathbf{AB}}) \vdash \\ &\qquad\qquad \mathbf{!}\, r(zs).(\boldsymbol{\nu} t)\,(\bar{t}\langle a\rangle \mid \mathbf{!}\, t(x).\overline{x}(\boldsymbol{\nu} z's').(z'.\bar{b}\langle zs\rangle + s'(x').\bar{t}\langle x'\rangle)) \end{aligned}$$

Note that my current type system (with the "delayed dependencies" extension) recognises that Geom is responsive, but due to $t$ calling itself, just produces $r_{\mathbf{R}}\triangleleft\bot$.

One useful extension improving the practicality of this work is known as *type reconstruction* in the literature (see f.i. [IK00]). In the current form of the type system, the programmer is required to provide channel types for all channels (both free and bound), although types for some channels can in some cases be inferred from the process. For instance, for bound names that never appear in object position of observable outputs, all receivers and senders are known, and the process types of their continuations could be used to construct the channel type.

Another future work direction is doing an actual software implementation. I did a Java-based implementation some time ago, prior to inclusion of branching and selection in the types. Choice makes some operations such as closure and detection of $\cong$-equivalence more difficult but there doesn't seem to be any serious difficulties. The closure uniqueness proof, Section A.1.5, gives hints for an implementation that doesn't rely on (inefficient) fixed point algorithms, and the proof of the normal form lemmas (Section A.1.6) suggests a way to simplify and compare behavioural statements.

# Bibliography

[AB08a] L. Acciai and M. Boreale. Responsiveness in process calculi. *Theoretical Computer Science*, 409(1):59–93, 2008.

[AB08b] L. Acciai and M. Boreale. Spatial and Behavioral Types in the Pi-Calculus. In *Proceedings of CONCUR'08*, volume 5201 of *LNCS*, pages 372–386. Springer, 2008.

[ABL03] R. M. Amadio, G. Boudol and C. Lhoussaine. The receptive distributed $\pi$-calculus. *ACM Transactions on Programming Languages and Systems*, 25(5):549–577, 2003.

[AG97] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security*, pages 36–47, New York, NY, USA, 1997. ACM.

[BDNN98] C. Bodei, P. Degano, F. Nielson and H. R. Nielson. Control Flow Analysis for the pi-calculus. In *CONCUR '98: Proceedings of the 9th International Conference on Concurrency Theory*, pages 84–98, London, UK, 1998. Springer-Verlag.

[BPV05] M. Baldamus, J. Parrow and B. Victor. A Fully Abstract Encoding of the $\pi$-Calculus with Data Terms. In *Proceedings of ICALP'05*, pages 1202–1213. Springer-Verlag, 2005.

[Cai04] L. Caires. Behavioral and Spatial Observations in a Logic for the $\pi$-Calculus. In *Proceedings of FOSSACS'04*, volume 2987 of *LNCS*. Springer, 2004.

[CC04] D. Cacciagrano and F. Corradini. Fairness in the pi-calculus. Technical Report, Dipartimenti di Informatica, Università di L'Aquila, 2004.

[CG90] N. Carriero and D. Gelernter. *How to write parallel programs: a first course.* MIT Press, Cambridge, MA, USA, 1990.

[DS06] Y. Deng and D. Sangiorgi. Ensuring termination by typability. *Information and Computation*, 204(7):1045–1082, 2006.

[GNR04] M. Gamboni, U. Nestmann and A. Ravara. What is TyCO, After All? Master's thesis, École Polytechnique Fédérale de Lausanne, 2004.

[Hen07] M. Hennessy. *A Distributed Pi-Calculus.* Cambridge University Press, New York, NY, USA, 2007.

[IK00] A. Igarashi and N. Kobayashi. Type reconstruction for linear $\pi$-calculus with I/O subtyping. *Information and Computation*, 161(1):1–44, 2000.

[IK01] A. Igarashi and N. Kobayashi. A generic type system for the Pi-calculus. *ACM SIGPLAN Notices*, 36(3):128–141, 2001.

[Kob02a] N. Kobayashi. A type system for lock-free processes. *Information and Computation*, 177(2):122–159, 2002.

[Kob02b] N. Kobayashi. Type systems for concurrent programs. In *Proceedings of UNU/IIST 10th Anniversary Colloquium*, volume 2757 of *LNCS*, pages 439–453. Springer, 2002.

[Kob08] N. Kobayashi. TyPiCal 1.6.2, 2008.

[KPT99] N. Kobayashi, B. C. Pierce and D. N. Turner. Linearity and the Pi-Calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999.

[KS08] N. Kobayashi and D. Sangiorgi. A Hybrid Type System for Lock-Freedom of Mobile Processes. In *Proceedings of CAV'08*, volume 5123 of *LNCS*, pages 80–93. Springer, 2008.

[KSS00] N. Kobayashi, S. Saito and E. Sumii. An Implicitly-Typed Deadlock-Free Process Calculus. In *Proceedings of CONCUR'00*, volume 1877, pages 489–503, 2000.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* Springer Verlag, 1980.

[Mil93] R. Milner. The Polyadic $\pi$-Calculus: A Tutorial. In *Logic and Algebra of Specification, Proceedings of the International NATO Summer School (Marktoberdorf, Germany, 1991)*, volume 94 of *NATO ASI Series F.* Springer, 1993.

[MPW92] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, 1992.

[Nes96] U. Nestmann. *On Determinacy and Nondeterminacy in Concurrent Programming.* PhD thesis, Universität Erlangen Nürnberg, 1996.

[Nes00] U. Nestmann. What Is a 'Good' Encoding of Guarded Choice? *Information and Computation*, 156:287–319, 2000. An extended abstract appeared in the *Proceedings of EXPRESS '97*, volume 7 of *ENTCS.*

[Par01] J. Parrow. An Introduction to the $\pi$-Calculus. In P. Bergstra and Smolka, eds, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.

[Par08] J. Parrow. Expressiveness of Process Algebras. *Electron. Notes Theor. Comput. Sci.*, 209:173–186, 2008.

[PS93] B. C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. In *Proceedings of LICS'93*, pages 376–385. IEEE Computer Society, 1993.

[PT00] B. C. Pierce and D. N. Turner. Pict: A Programming Language Based on the Pi-Calculus. In G. Plotkin, C. Stirling and M. Tofte, eds, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[San93] D. Sangiorgi. From pi-Calculus to Higher-Order pi-Calculus - and Back. In *TAPSOFT '93: Proceedings of the International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, pages 151–166, London, UK, 1993. Springer-Verlag.

[San98] D. Sangiorgi. An Interpretation of Typed Objects into Typed $\pi$-Calculus. *Information and Computation*, 143(1):34–73, 1998. Earlier version published as Rapport de Recherche RR-3000, INRIA Sophia-Antipolis, August 1996, and presented at FOOL 3.

[San99] D. Sangiorgi. The Name Discipline of Uniform Receptiveness. *Theoretical Computer Science*, 221(1–2):457–493, 1999.

[SW01] D. Sangiorgi and D. Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[Vas94] V. T. Vasconcelos. Typed Concurrent Objects. In *8th European Conference on Object-Oriented Programming*, volume 821 of *Lecture Notes in Computer Science*, pages 100–117. Springer-Verlag, July 1994.

[YBH04] N. Yoshida, M. Berger and K. Honda. Strong normalisation in the $\pi$-calculus. *Information and Computation*, 191(2):145–202, 2004.

# Appendix A

# Proofs

We prove in this section a number of important properties of the type system, such as subject reduction, type safety and type soundness.

## A.1 Proofs for Sections 3, 4 and 5

### A.1.1 Auxiliary Lemmas

All operators used in behavioural statements are idempotent and distributive, which lets us prove the following property:

**Lemma A.1.1 (Nesting Elimination Lemma)** *Let $C[\cdot]$ and $C'[\cdot]$ be two behavioural contexts and $\varepsilon$ a behavioural statement. Then*

$$C[C'[C[\varepsilon]]] \cong C[C'[\varepsilon]]$$

*Proof*

First consider the case $C[\cdot] = \varepsilon_0 \vee [\cdot]$.

Repeatedly using the laws $\varepsilon_0 \vee (\varepsilon_1 \wedge \varepsilon_2) \cong (\varepsilon_0 \vee \varepsilon_1) \wedge (\varepsilon_0 \vee \varepsilon_2)$ and $\varepsilon_0 \vee (\varepsilon_1 \vee \varepsilon_2) \cong (\varepsilon_0 \vee \varepsilon_1) \vee (\varepsilon_0 \vee \varepsilon_2)$, we transform $C[C'[C[\varepsilon]]]$ to $C'_0[\varepsilon_0 \vee C[\varepsilon]]$, where $C'_0[\ ]$ is $C'[\ ]$ with $\varepsilon_0 \vee$ prefixing every individual term except the hole. Substituting $C[\cdot]$ with its definition we get $C'_0[\varepsilon_0 \vee \varepsilon_0 \vee \varepsilon]$ which is $\cong$-equivalent to $C'_0[\varepsilon_0 \vee \varepsilon]$. Reversing the "$\varepsilon_0$-injection" done above, we obtain $\varepsilon_0 \vee C'[\varepsilon]$, i.e. $C[C'[\varepsilon]]$.

The proof for $C[\cdot] = \varepsilon_0 \wedge [\cdot]$ is identical but using $\wedge$ instead of $\vee$.

Any behavioural context can be written as a composition of contexts of the above two forms, so let $C[\cdot] = C_1[C_2[\cdots C_n[\cdot]\cdots]]$. The statement being considered is

$$C_1[C_2[\cdots C_n[C'[C_1[C_2[\cdots C_n[\varepsilon]\cdots]]]]\cdots]]$$

Using the above base case it can be reduced to

$$C_1[C_2[\cdots C_n[C'[C_2[\cdots C_n[\varepsilon]\cdots]]]\cdots]]$$

As $\cong$ is a congruence, the inner $C_2[\cdot]$ can similarly be dropped, and so can all the others. □

**Lemma A.1.2 (Weakening Conserves Structure)** *Let $\Delta = \Delta_1 \wedge \Delta_2$, and $\Delta' \succeq \Delta$. Then $\Delta' \cong \Delta'_1 \wedge \Delta'_2$ with $\Delta'_i \succeq \Delta_i$ for both $i$. The same property holds for $\vee$ instead of $\wedge$ or $\preceq$ instead of $\succeq$.*

*Proof* Rule $\eta_1 \wedge \eta_2 \preceq \eta_1$ can be written $\eta_1 \wedge \eta_2 \preceq \eta_1 \wedge \top$ (and note that $\eta_2 \preceq \top$) and $\eta_1 \preceq \eta_1 \vee \eta_2$ can be written $\eta_1 \vee \bot \preceq \eta_1 \vee \eta_2$.

The remaining rules in Definition 3.6.1 either are already in the required form, or actually define $\cong$, in which case one can simply set $\Delta_i' = \Delta_i$ for both $i$. □ The following lemma states that a labelled transition can be split into two phases, one that may perform up to two branchings (by replacing a sum by one of its elements) and the second does the actual transition. This makes it possible to split proofs similarly. Note that this lemma only holds because our process calculus doesn't include replicated sums such as $\mathbf{!}\,(a+b)$ (but accepts the strongly bisimilar $\mathbf{!}\,a|\mathbf{!}\,b$).

**Lemma A.1.3 (Branching Transition)** *Let $P \xrightarrow{\mu} P'$ be a transition. Then there is a process $\hat{P}$ such that*

- *$\hat{P}$ is obtained from $P$ by replacing at most two sums $\sum_{i \in I} G_i.P_i$ by $G_{\hat{\imath}}.P_{\hat{\imath}}$ for some $\hat{\imath} \in I$.*
- *$\hat{P} \xrightarrow{\mu} P'$, without using the* (Sum) *rule from the labelled transition system.*

The following lemma, whose proof is omitted, will be helpful in many proofs:

**Lemma A.1.4 (Structural Lemma)** *Let $P$ be a process and $P \xrightarrow{\mu} P'$ where $\mathsf{sub}(\mu) = p$ and* (Sum) *was not used. Then $P$ is of the following form:*

$$P \equiv (\boldsymbol{\nu}\tilde{z})\ \big(Q \mid G.R\big)$$

*where $\mathrm{n}(p) \notin \tilde{z}$ and $\mathsf{sub}(G) = p$, and, if $\mu$ is an output, $\mathsf{obj}(G) \cap (\mathrm{bn}(G) \cup \tilde{z}) = \mathrm{bn}(\mu)$. For $P'$, either (when $\#(G) = 1$)*

$$P' \equiv (\boldsymbol{\nu}\tilde{z} \setminus \mathrm{bn}(\mu))\ \Big(Q \mid R\{^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}\Big)$$

*or (when $\#(G) = \omega$)*

$$P' \equiv (\boldsymbol{\nu}\tilde{z} \setminus \mathrm{bn}(\mu))\ \Big(Q \mid G.R \mid R\{^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}\Big)\,.$$

*Now let instead $P \xrightarrow{\tau} P'$, still not using* (Sum)*. Then*

$$P \equiv (\boldsymbol{\nu}\tilde{z})\ \big(Q \mid G.R \mid G'.R'\big)$$

*where there is a name $a$ s.t. $\mathsf{sub}(G) = a$ and $\mathsf{sub}(G') = \bar{a}$. Similarly to $\mu \neq \tau$ there are four cases for $P'$, depending on $\#(G)$ and $\#(G')$, but we only show the one where both are 1:*

$$P' \equiv (\boldsymbol{\nu}\tilde{z} \cup \mathrm{bn}(G'))\ \Big(Q \mid\ \Big(R\{^{\mathsf{obj}(G')}/_{\mathsf{obj}(G)}\}\Big)\ \mid R'\Big)\,.$$

Finally, the following lemma gives a few useful properties of process type operators:

**Lemma A.1.5** *Let $\Gamma_1$ and $\Gamma_2$ be process types, $m_1$ and $m_2$ multiplicities.*

- *$(m_1 + m_2) - m_2 \succeq m_1$.*

- *If* $\Gamma_1 \odot \Gamma_2$ *is well defined then* $(\Gamma_1 \odot \Gamma_2) \setminus \Gamma_2 \succeq \Gamma_1$
- *If* $\Gamma_1 \odot \Gamma_2$ *is well defined and* $\Gamma_1' \succeq \Gamma_1$ *then* $\Gamma_1' \odot \Gamma_2$ *is also well defined and* $\Gamma_1' \odot \Gamma_2 \succeq \Gamma_1 \odot \Gamma_2$
- *Let* $\Gamma \vdash P$*,* $\Gamma' \vdash P'$ *with* $\Gamma' \succeq \Gamma$*. If* $\Gamma_2 \vdash C[P]$*, using* $\Gamma \vdash P$ *in the derivation, then there is* $\Gamma_2$ *with* $\Gamma_2' \vdash C[P']$ *(using* $\Gamma' \vdash P'$ *in the derivation) and* $\Gamma_2' \succeq \Gamma_2$*.*

## A.1.2 Properties of $\cong$ (Lemma 3.6.2)

Up to $\cong$, $\bot$ is neutral for $\vee$ and absorbent for $\wedge$. $\top$ is absorbent for $\vee$ and neutral for $\wedge$.

We show $\bot$ is neutral for $\vee$ ($\top$ being neutral for $\wedge$ is similar).
By $\eta_1 \preceq \eta_1 \vee \eta_2$ we have $\eta \vee \bot \succeq \eta$.
By $\bot \preceq \eta$, $\eta \vee \bot \preceq \eta \vee \eta$ which (as $\vee$ is idempotent) implies $\eta \vee \bot \preceq \eta$.
We now show $\top$ is absorbent for $\vee$:
By $\eta_1 \preceq \eta_1 \vee \eta_2$, $\eta \vee \top \succeq \top$
By $\eta \preceq \top$, $\eta \vee \top \preceq \top$.

## A.1.3 Composition Properties (Lemma 3.9.4)

The $+$ operator on multiplicities is commutative as can be seen in Definition 3.8.1. It has a neutral element 0 as stated in the same definition, and is associative (one can easily see that $a_1 + (a_2 + a_3)$ is $\star$ if two or more $a_i$ are non-zero, and is $a_i$ if both $a_j$ with $j \neq i$ are zero, so rotating the $a_i$ preserves the result).

The behavioural statement operators $\vee$ and $\wedge$ are commutative up to $\cong$ (Definition 3.6.1).

**Commutativity of behavioural statement composition** The $\Delta_1 \odot \Delta_2 \cong \Delta_2 \odot \Delta_1$ equivalence is proven by structural induction on $\Delta_1$ and $\Delta_2$. One of the cases is: Assume $\Theta_i \odot \Delta_2 \cong \Delta_2 \odot \Theta_i$ for both $i \in \{1, 2\}$. Then $(\Theta_1 \wedge \Theta_2) \odot \Delta_2$ is $\cong$ to ($\odot$ being a logical homomorphism) $(\Theta_1 \odot \Delta_2) \wedge (\Theta_2 \odot \Delta_2)$ whish is $\cong$ to (by induction hypothesis) $(\Delta_2 \odot \Theta_1) \wedge (\Delta_2 \odot \Theta_2)$, $\cong$ to ($\odot$ being a logical homomorphism) $\Delta_2 \odot (\Theta_1 \wedge \Theta_2)$. Other “step” cases are similar. The base cases enumerated in Definition 5.1.1 follow from $+$, $\wedge$ and $\vee$ being commutative.

**Associativity of behavioural statement composition** $\Delta_1 \odot (\Delta_2 \odot \Delta_3) \cong (\Delta_1 \odot \Delta_2) \odot \Delta_3)$ is again proven by structural induction on all three statements. The step cases are much similar to the above, exploiting $\odot$ being a logical homomorphism and the distributivity rules of $\cong$ to decompose the product, apply the induction hypothesis and recompose the resulting terms. For the induction base case, assume all three $\Delta_i$ are of the form $p^m$ and $\gamma \triangleleft \varepsilon$. Note that if they are not all dependency statements of the same resource $\gamma$, or all multiplicities of the same port $p$, rule 4 of Definition 5.1.1 will apply and return $\top$ no matter in which order the $\Delta_i$ are composed. Otherwise, the three remaining base cases corresponding to the first three points of Definition 5.1.1 satisfy associativity as a consequence of $+$, $\vee$ and $\wedge$ being associative up to $\cong$.

As a corollary of Lemma 5.1.2, $\top$ is a neutral element of $\odot$ when Convention 4.2.2 applies.

We may now lift the above results to prove the Lemma itself.

**Proof of the Lemma** By commutativity of $\wedge$ and $\odot$ on behavioural statements,

$$(\Sigma_1 \wedge \Sigma_2\,;\, \Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2} \blacktriangleleft (\Xi_{\mathrm{E}1} \setminus \Xi_{\mathrm{L}2}) \wedge (\Xi_{\mathrm{E}2} \setminus \Xi_{\mathrm{L}1})) \cong (\Sigma_2 \wedge \Sigma_1\,;\, \Xi_{\mathrm{L}2} \odot \Xi_{\mathrm{L}1} \blacktriangleleft (\Xi_{\mathrm{E}2} \setminus \Xi_{\mathrm{L}1}) \wedge (\Xi_{\mathrm{E}1} \setminus \Xi_{\mathrm{L}2})) \quad \text{(A.1)}$$

As closure and removal of non-observable dependencies commute with $\cong$ (Lemma 3.11.5), $\odot$ on process types is commutative.

$(\varnothing; \top \blacktriangleleft \top)$ is a neutral element: Let $\Delta$ be any behavioural statement. Then $\Delta \setminus \top = \Delta$, and $\top \setminus \Delta = \top$, both consequences of point 4 in Definition 3.9.1. Then:

$$\begin{aligned}(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \odot (\varnothing; \top \blacktriangleleft \top) &= (\Sigma \wedge \varnothing\,;\, \Xi_{\mathrm{L}} \odot \top \blacktriangleleft (\Xi_{\mathrm{E}} \setminus \top) \wedge (\top \setminus \Xi_{\mathrm{L}}))\\ &= (\Sigma \cup \varnothing; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}} \wedge \top)\\ &\cong (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})\end{aligned}$$

Again, the remaining points of Definition 4.2.6 commute with $\cong$ so we are done.

Regarding associativity, let $\Gamma_i = (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i})$ for $i \in \{1,2,3\}$, $\Gamma = (\Gamma_1 \odot \Gamma_2) \odot \Gamma_3$ and $\Gamma' = (\Gamma_3 \odot \Gamma_2) \odot \Gamma_1$. We show that $\Gamma \cong \Gamma'$.

Let $\Xi_{\mathrm{L}12}$ be close $(\Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2})$ without resources not observable in $\Xi_{\mathrm{E}1} \setminus \Xi_{\mathrm{L}2} \wedge \Xi_{\mathrm{E}2} \setminus \Xi_{\mathrm{L}1}$. Then $\Gamma_1 \odot \Gamma_2 = (\Sigma_1 \wedge \Sigma_2; \Xi_{\mathrm{L}12} \blacktriangleleft \Xi_{\mathrm{E}1} \setminus \Xi_{\mathrm{L}2} \wedge \Xi_{\mathrm{E}2} \setminus \Xi_{\mathrm{L}1})$. The first step (from Definition 4.2.6) for computing $\Gamma$ is then

$$((\Sigma_1 \wedge \Sigma_2) \wedge \Sigma_3; \Xi_{\mathrm{L}12} \odot \Xi_{\mathrm{L}3} \blacktriangleleft \Xi_{\mathrm{E}3} \setminus \Xi_{\mathrm{L}12} \;\wedge\; (\Xi_{\mathrm{E}1} \setminus \Xi_{\mathrm{L}2} \wedge \Xi_{\mathrm{E}2} \setminus \Xi_{\mathrm{L}1}) \setminus \Xi_{\mathrm{E}3})\,.$$

The following property helps computing the environment component:

$$\forall \Delta, \Delta_1, \Delta_2 : \big(\Delta_1 \hookrightarrow \Delta_2\big) \;\Rightarrow\; \big(\Delta \setminus \Delta_1 \cong \Delta \setminus \Delta_2\big) \qquad \text{(A.2)}$$

We omit the proof but essentially, dependency reduction preserves the only parts of $\Delta_1$ that matter when computing the subtraction $\Delta \setminus \Delta_i$. In particular, $\Xi_{\mathrm{E}3} \setminus \Xi_{\mathrm{L}12} \cong \Xi_{\mathrm{E}2} \setminus (\Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2})$.

Secondly,

$$\forall \Delta_1, \Delta_2, \Delta_3 : \Delta_1 \setminus (\Delta_2 \odot \Delta_2) \cong (\Delta_1 \setminus \Delta_2) \setminus \Delta_3$$

which is proved by "lifting up" the corresponding equality $m_1 - (m_2 + m_2) = (m_1 - m_2) - m_3$ on multiplicities.

The environment component, as $\setminus$ distributes over $\wedge$ (Definition 3.9.1), is therefore $\cong$-equivalent to

$$\Xi_{\mathrm{E}3} \setminus (\Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2}) \wedge \Xi_{\mathrm{E}1} \setminus (\Xi_{\mathrm{L}2} \odot \Xi_{\mathrm{L}3}) \wedge \Xi_{\mathrm{E}2} \setminus (\Xi_{\mathrm{L}3} \odot \Xi_{\mathrm{L}1})$$

for which it is easy to see that swapping 3 and 1 indexes yields an equivalent statement.

Step two for computing $\Gamma$ is doing the closure of the local statement $\Xi_{\mathrm{L}12} \odot \Xi_{\mathrm{L}3}$. By closure uniqueness,

$$\mathrm{close}\,(\mathrm{close}\,(\Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2}) \odot \Xi_{\mathrm{L}3}) \cong \mathrm{close}\,(\Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2} \odot \Xi_{\mathrm{L}3})$$

in which, again, swapping 1 and 3 yields an equivalent statement.

As far as step three is concerned, dropping non-observable resources commutes with statement equivalence so we are done.

### A.1.4 Simple Correctness and Structural Equivalence (L. 3.12.2)

This lemma has two parts that can be proven independently:

1. simple correctness is preserved by structural congruence
2. simple correctness is preserved by type equivalence

The proof of part 1 relies on two elementary properties of structural congruence whose proof is omitted: $\equiv$ is a strong bisimulation ($Q \equiv P \xrightarrow{\mu} P'$ implies $\exists Q' : Q \xrightarrow{\mu} Q' \equiv P'$) and preserves the set of free names ($P \equiv Q$ implies $\mathrm{fn}(P) = \mathrm{fn}(Q)$).

Let $\Gamma \models_{\#} P$ and $Q \equiv P$. We show that $\Gamma \models_{\#} Q$ as well. Point 1 of Definition 3.12.1 is an immediate consequence of $\Gamma \models_{\#} P$ and $\equiv$ preserving the set of free names.

Point 2 of Definition 3.12.1 is an immediate consequence of $\Gamma \models_{\#} P$ and $\equiv$ being a bisimulation, keeping for $Q$ the same $\Gamma_+$ that was used for $P$.

Point 3 of Definition 3.12.1 is done by inspecting a proof of $\equiv$ being a bisimulation: No application of (Rep) is ever added or removed when transforming $P \xrightarrow{\mu} P'$ to $Q \xrightarrow{\mu} Q'$. Concerning uniqueness of the $\mu$ transition: the set of top-level guards, and whether their subject port is free is preserved by $\equiv$.

We now proceed to part 2 of this proof (type equivalence preserves simple correctness). Let $(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'; P')$ be a transition sequence where $\Gamma \models_{\#} P$, and let $\Theta \cong \Gamma$. As the transition operator commutes with $\cong$-equivalence, there is $(\Theta; P) \xrightarrow{\tilde{\mu}} (\Theta'; P')$ with $\Theta' \cong \Gamma'$.

Property 1 from Definition 3.12.1 is satisfied as the channel types in $\Gamma'$ and $\Theta'$ must be equal, by definition of $\cong$.

For property number 2, there is a set of ports $\tilde{p}$ whose environment multiplicity got raised to $\star$ in $\Gamma_+$, and let $\Theta_+$ be equal to $\Theta'$ but setting environment multiplicities of $\tilde{p}$ to $\star$. Again, as $\cong$ commutes with $\wr$, keeping the same $\mu'$ as with $\Gamma_+$, $\Theta_+ \wr \mu'$ is well defined.

Property number 3 is satisfied because the multiplicity of a port is preserved by type equivalence.

### A.1.5 Closure Uniqueness (Lemma 4.2.4)

We proceed in increasing generality, by first focusing on special cases. Let:

$$\Delta = \bigwedge_{i \in I} \gamma_i \triangleleft \varepsilon_i \tag{A.3}$$

where $\gamma_i \neq \gamma_{i'}$ for any distinct $i$ and $i'$. We only consider points 1, 2 and 4 from Definition 5.1.3 for the time being. The following definition allows to merge the first two rules:

**Notation A.1.6 (Alternative Operator)** *Let $p_k$ be a resource and $\varepsilon$ a dependency. Then $p_k * \varepsilon$ is equal to $p_k \vee \varepsilon$ if $k = \mathbf{A}$, and to $p_k \wedge \varepsilon$ if $k = \mathbf{R}$.*

We write $\Delta \setminus \tilde{\alpha}$ to mean $(\bigwedge_{i \in I : \gamma_i \notin \tilde{\alpha}} \gamma_i \lhd \varepsilon_i) \wedge (\bigwedge_{\alpha \in \tilde{\alpha}} \alpha \lhd \bot)$, and $\hat{\Delta}(\gamma_i)$ is $\hat{\varepsilon}_i$, $\gamma_i$'s dependencies in $\hat{\Delta}$. The following definition can be used to construct a closure explicitly:

**Definition A.1.7 ($\Delta$-Closure)** *A* $\Delta$-closure *of a statement* $\Theta$ *(typically chosen equal to* $\Delta$*) is a statement* $\mathrm{close}_{(\Delta)}(\Theta) = \Theta'$ *inductively constructed as follows:*

1. $\mathrm{close}_{(\Delta)}(\top) \overset{\text{def}}{=} \top$ *and* $\mathrm{close}_{(\Delta)}(\bot) \overset{\text{def}}{=} \bot$
2. $\mathrm{close}_{(\Delta)}(\gamma \lhd \varepsilon) \overset{\text{def}}{=} \gamma \lhd (\mathrm{close}_{(\Delta \setminus \gamma)}(\varepsilon))$
3. $\mathrm{close}_{(\Delta)}(\gamma) \overset{\text{def}}{=} \gamma * \mathrm{close}_{(\Delta \setminus \gamma)}(\Delta(\varepsilon))$.
4. $\mathrm{close}_{(\bigwedge_{i \in I} \gamma_i \lhd \varepsilon_i)}(\gamma) \overset{\text{def}}{=} \gamma$ *if* $\nexists i \in I : \gamma_i = \gamma$.
5. $\mathrm{close}_{(\Delta)}(\Delta_1 \wedge \Delta_2) \overset{\text{def}}{=} \mathrm{close}_{(\Delta)}(\Delta_1) \wedge \mathrm{close}_{(\Delta)}(\Delta_2)$.

It is easily seen by induction on the number of symbols appearing in the representation of $\Delta$ plus the number of statements in $\Theta$ that do not depend on $\bot$, that the above procedure terminates after a finite number of steps.

We will now show that $\mathrm{close}_{(\Delta)}(\Delta) = \mathrm{close}(\Delta)$.

Let $\hat{\Delta} = \mathrm{close}_{(\Delta)}(\Delta)$. Then any $\Delta'$ such that $\hat{\Delta} \hookrightarrow \Delta'$ satisfies $\hat{\Delta} \cong \Delta'$. In other words, for all distinct $j$ and $k$:

$$\varepsilon'_k \overset{\text{def}}{=} \hat{\varepsilon}_k\{^{\gamma_j * (\hat{\varepsilon}_j\{^{\bot}/_{\gamma_k}\})}/_{\gamma_j}\} \cong \hat{\varepsilon}_k \tag{A.4}$$

By construction, every $\gamma_i$ appearing on the rhs of a $\lhd$ operator occurs as $\gamma_i * \mathrm{close}_{(\Delta \setminus \tilde{\gamma})}(\varepsilon_i)$ where $\tilde{\gamma}$ is the set of all resources "wrapping" that statement (including $\gamma_i$). Moreover, within a statement $\gamma_i \lhd \varepsilon$ or $\gamma_i * \varepsilon$, any $\gamma_i$ appearing in $\varepsilon$ occurs as $\gamma_i * \bot$.

Assume w.l.o.g. that $\gamma_j$ appears exactly once in $\hat{\varepsilon}_k$ (if it never appears then $\hat{\varepsilon}_k = \varepsilon'_k$, and if it appears more than once, simply repeat the construction below that many times). We write $C_{\mathbf{k}}[\cdot]$ for the unique behavioural context (a behavioural statement with one hole $[\cdot]$) such that $\hat{\varepsilon}_k = C_{\mathbf{k}}[\gamma_j * \mathrm{close}_{(\Delta \setminus \tilde{\gamma})}(\varepsilon_j)]$. Applying the substitution in (A.4) we get

$$\varepsilon'_k = C_{\mathbf{k}}[\gamma_j * \big(\mathrm{close}_{(\Delta \setminus \tilde{\gamma})}(\varepsilon_j)\ ,\ \mathrm{close}_{(\Delta \setminus \{\gamma_j, \gamma_k\})}(\varepsilon_j)\big)] \tag{A.5}$$

Now assume w.l.o.g. that there is exactly one $\gamma_l \in \tilde{\gamma}$ that appears in $\hat{\varepsilon}_j$, and moreover that $\gamma_l$ appears exactly once in $\hat{\varepsilon}_j$. (Again, if there's more than one occurence of a resource from $\tilde{\gamma}$ in $\hat{\varepsilon}_j$, then all of them can be individually transformed as described below. If there's none, $\mathrm{close}_{(\Delta \setminus \tilde{\gamma})}(\varepsilon_j) = \mathrm{close}_{(\Delta \setminus \gamma_j \gamma_k)}(\varepsilon_j)$, and $\varepsilon'_k \cong \hat{\varepsilon}_k$ follows from $\gamma * (\varepsilon, \varepsilon)$ being either $\gamma \vee \varepsilon \vee \varepsilon$ or $\gamma \wedge \varepsilon \wedge \varepsilon$, that both reduce to $\gamma * \varepsilon$.) Let $C_{\mathbf{j}}[\cdot]$ by the only behavioural context such that

$$\hat{\varepsilon}_j = C_{\mathbf{j}}[\gamma_l * \mathrm{close}_{(\Delta \setminus \tilde{\gamma}')}(\varepsilon_l)] \tag{A.6}$$

for some $\tilde{\gamma}'$ with $\gamma_j \in \tilde{\gamma}'$.

The complete dependency chain obtained above can be seen in the following diagram, where $C_{\mathbf{k}}[\cdot]$ is the composition of the two arrows from $\gamma_k$ to $\gamma_j$, and $C_{\mathbf{j}}[\cdot]$ is represented by the arrow going back from $\gamma_j$ to $\gamma_l$.

$$\begin{tikzcd} \boxed{\gamma_k} \arrow[r] & \boxed{\gamma_l} \arrow[r] & \boxed{\gamma_j} \arrow[l, bend right=60] \end{tikzcd}$$

As $\gamma_l \in \tilde{\gamma}$, the context $C_{\mathbf{k}}[\cdot]$ can uniquely be split into $C^0_{\mathbf{k}}[\cdot]$ and $C_{\mathbf{l}}[\cdot]$ (corresponding to the two horizontal arrows in the diagram) so that $C_{\mathbf{k}}[\cdot] = C^0_{\mathbf{k}}[C_{\mathbf{l}}[\cdot]]$, and $\text{close}_{(\Delta\setminus\tilde{\gamma}')}(\varepsilon_l) = C_{\mathbf{l}}[\gamma_j * \bot]$ (note that $\gamma_j \in \tilde{\gamma}'$ implies $\tilde{\gamma}'(\gamma_j) = \bot$).

Composing (A.4) and (A.6) we get

$$\varepsilon'_k \cong C_{\mathbf{k}}[\gamma_j * \big(C_{\mathbf{j}}[\gamma_l * \bot]\ ,\ C_{\mathbf{j}}[\gamma_l * C_{\mathbf{l}}[\gamma_j * \bot]]\big)]$$

Splitting $C_{\mathbf{k}}[\cdot]$:

$$\varepsilon'_k \cong C^0_{\mathbf{k}}[C_{\mathbf{l}}[\gamma_j * \big(C_{\mathbf{j}}[\gamma_l * \bot]\ ,\ C_{\mathbf{j}}[\gamma_l * C_{\mathbf{l}}[\gamma_j * \bot]]\big)]]$$

Applying the Nesting Elimination Lemma (A.1.1) with "$C_{\mathbf{l}}[\gamma_j * [\cdot]]$" for $C[\cdot]$, this becomes

$$\varepsilon'_k \cong C^0_{\mathbf{k}}[C_{\mathbf{l}}[\gamma_j * \big(C_{\mathbf{j}}[\gamma_l * \bot]\ ,\ C_{\mathbf{j}}[\gamma_l * \bot]\big)]]$$

By idempotence, and reuniting $C^0_{\mathbf{k}}[C[\cdot]]$ to $C_{\mathbf{k}}[\cdot]$ we get $\varepsilon'_k \cong C_{\mathbf{k}}[\gamma_j * \big(C_{\mathbf{j}}[\gamma_l * \bot]\big)] = \hat{\varepsilon}_k$, as required.

This completes the proof that $\Delta' = \text{close}_{(\Delta)}(\Delta)$ is a closure. We still need to show that it is the only closure, i.e. any closure of $\Delta$ is $\cong$-equivalent to $\Delta'$.

Let $\Delta \hookrightarrow \Delta''$ be s.t. $\Delta'' \hookrightarrow \Delta'''$ implies $\Delta'' \cong \Delta'''$ for all $\Delta'''$.

By the definition of $\hookrightarrow$, $\Delta''$ can be obtained from $\Delta$ by, a certain number of times, replacing $\gamma_i$ by $\gamma_i * \varepsilon_i$. (Technically an individual application of a rule in 5.1.3 introduces some $\varepsilon'_i$ not necessarily equal to $\varepsilon_i$ but as $\varepsilon'_i$ was itself obtained from $\varepsilon_i$ by applying similar transformations, this description is correct).

A resource occurrence $\gamma_j$ in a statement is said "bare" if it is neither followed by the $*$-operator nor contained in the $\varepsilon$ of a statement $\gamma_j * \varepsilon$.

A bare occurrences of a resource $\gamma_j$ can be "completed" by applying Definition A.1.6 to replace all $\gamma_j$ in the offending statement by $\gamma_j * (\Delta''(\gamma_j)\{^{\bot}/_{\gamma_k}\})$. Repeating this procedure as many times as required produces a statement $\Delta'''$ that has no bare resource occurrences, and that satisfies $\Delta'' \hookrightarrow \Delta'''$. As $\Delta''$ was assumed to be a closure, $\Delta'' \cong \Delta'''$. Nested resource developments ($\gamma_i * \varepsilon$ where $\varepsilon$ contains $\gamma_i * \varepsilon'$ for some $\varepsilon'$ can be reduced as shown above (replacing $\gamma_i * \varepsilon'$ by $\gamma_i * \bot$), resulting in $\Delta''' \cong \text{close}_{(\Delta)}(\Delta)$, as required.

## A.1.6 Normal Form (Lemma 4.2.13)

We only prove point 1 as point 2 is similar (note that the direction of the relation is inversed because adding terms to a disjunction makes it weaker, while adding terms to a conjunction makes it stronger).

Let $\{\varepsilon_i\}_i$ and $\{\varepsilon_j\}_j$ be sets of dependencies as in the Lemma statement. For all $j \in J$, let $\varepsilon'_j = \varepsilon_i$ such that $\varepsilon'_j \preceq \varepsilon_j$. As $\preceq$ is a congruence relation we have

$$\bigvee_{j\in J} \varepsilon'_j \preceq \bigvee_{j\in J} \varepsilon_j \tag{A.7}$$

By idempotence, multiple $\varepsilon'_j$ equal to the same $\varepsilon_i$ can be replaced by a single one, so we have

$$\bigvee_{i\in I_0} \varepsilon_i \cong \bigvee_{j\in J} \varepsilon'_j \tag{A.8}$$

where $I_0 = \{i \in I \ : \ \exists j \in J : \varepsilon'_j = \varepsilon_i\}$. Applying the $\varepsilon \vee \varepsilon' \preceq \varepsilon$ rule we get

$$\bigvee_{i\in I} \varepsilon_i \preceq \bigvee_{i\in I_0} \varepsilon_i \tag{A.9}$$

as $I_0 \subseteq I$. Composing the three above relations we have the desired inequality.

### A.1.7 Composition of Disjoint Statements (Lemma 5.1.2)

According to Convention 4.2.2, $\Xi$ and $\Xi'$ can be respectively written as $\Xi \wedge \bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\triangleleft\top$ and $\Xi' \wedge \bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_i\mathbf{R}\triangleleft\top$, where $\{p_i\}_{i\in M}$ is the set of ports that have a multiplicity specified in $\Xi'$, $\{p_i\}_{i\in R}$ is the set of ports whose responsiveness appear in $\Xi'$ on the lhs of a dependency "$\triangleleft$" (and the other way round for $M'$ and $R'$).

In other words,

$$\Xi\odot\Xi' \overset{\text{def}}{=} \Xi_r = \big(\Xi\wedge \bigwedge_{i\in M} {p_i}^0\wedge\bigwedge_{i\in R} p_i\mathbf{R}\triangleleft\top\big) \odot \big(\Xi'\wedge \bigwedge_{i\in M'} {p_i}^0\wedge \bigwedge_{i\in R'} p_i\mathbf{R}\triangleleft\top\big). \tag{A.10}$$

From the $\Xi_r$ written in (A.10) onwards, until the end of this proof, Convention 4.2.2 no longer applies, in particular $\Xi \odot \Xi'$ appearing in the development below is not considered to have "hidden" resources.

As $\odot$ is a logical homomorphism,

$$\begin{aligned}\Xi_r \cong (\Xi \odot \Xi') \wedge \big(( \bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\triangleleft\top) \odot \Xi'\big) \wedge \\ \big(\Xi \odot ( \bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_i\mathbf{R}\triangleleft\top)\big) \wedge \\ \big(( \bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\triangleleft\top) \odot ( \bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_i\mathbf{R}\triangleleft\top)\big)\end{aligned} \tag{A.11}$$

Similarly developing the $\Xi \odot \Xi'$ expression down to its individual terms and applying point 5 of the Definition to all of them we obtain a behavioural statement using only $\top$, $\vee$ and $\wedge$, i.e. $\Xi \odot \Xi' \cong \top$. The same applies to the fourth term: as $\Xi$ and $\Xi'$ have no common resources and $M$, $R$, $M'$ and $R'$ index resources in $\Xi$ and $\Xi'$, we get $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\triangleleft\top) \odot (\bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_i\mathbf{R}\triangleleft\top) \cong \top$. We are left with

$$\big(( \bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\triangleleft\top) \odot \Xi'\big) \wedge \big(\Xi \odot ( \bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_i\mathbf{R}\triangleleft\top)\big).$$

We concentrate on the left factor (the right one is similar). Let's distribute $\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\triangleleft\top$ into $\Xi'$ using $\odot$'s logical homomorphism. We obtain a behavioural statement equal to $\Xi'$ where every atomic statement $p^m$ or $\gamma\triangleleft\varepsilon$ got replaced by $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\lhd\top) \odot p^m$ or $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\lhd\top) \odot (\gamma\lhd\varepsilon)$, respectively. In the first case, $\exists i \in M$ s.t. $p_i = p$, so it is equal to

$$\bigwedge_{i\in M;\ p_i\neq p} ({p_i}^0 \odot p^m) \ \wedge\ (p^0 \odot p^m) \ \wedge\ \bigwedge_{i\in R} (p_i\mathbf{R}\lhd\top \odot p^m)$$

i.e. (using point 1 of the Definition on the middle, and 5 for the rest)

$$\bigwedge_{i\in M;\ p_i\neq p} \top \ \wedge\ p^{0+m} \ \wedge\ \bigwedge_{i\in R} \top \ \cong p^m$$

In the second case, for a responsiveness statement, $\exists i \in R$ s.t. $\gamma = p_i\mathbf{R}$, so it is equal to

$$\bigwedge_{i\in M} ({p_i}^0 \odot (\gamma\lhd\varepsilon)) \ \wedge \bigwedge_{i\in R;\ p_i\mathbf{R}\neq\gamma} (p_i\mathbf{R}\lhd\top \odot (\gamma\lhd\varepsilon)) \ \wedge\ (\gamma\lhd\top \odot (\gamma\lhd\varepsilon))$$

i.e. (using point 2 of the Definition on the right, and 5 for the rest)

$$\bigwedge_{i\in M} \top \ \wedge \bigwedge_{i\in R;\ p_i\mathbf{R}\neq\gamma} \top \ \wedge\ ((\gamma\lhd\top) \wedge (\gamma\lhd\varepsilon)) \cong \gamma\lhd\varepsilon$$

Finally, for an activeness statement, noting that $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\lhd\top) \cong (\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\lhd\top) \wedge \top \cong (\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\lhd\top) \wedge (\gamma\lhd\bot)$:

$$\bigwedge_{i\in M} ({p_i}^0 \odot (\gamma\lhd\varepsilon)) \ \wedge\ \bigwedge_{i\in R} (p_i\mathbf{R}\lhd\top \odot (\gamma\lhd\varepsilon)) \ \wedge\ (\gamma\lhd\bot \odot (\gamma\lhd\varepsilon))$$

i.e. (using point 3 of the Definition on the right, and 5 for the rest)

$$\bigwedge_{i\in M} \top \ \wedge \bigwedge_{i\in R;\ p_i\mathbf{R}\neq\gamma} \top \ \wedge\ ((\gamma\lhd\bot) \vee (\gamma\lhd\varepsilon)) \cong \gamma\lhd\varepsilon$$

We conclude that $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_i\mathbf{R}\lhd\top) \odot \Xi' \cong \Xi'$, and similarly $\Xi \odot (\bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_i\mathbf{R}\lhd\top) \cong \Xi$, so (A.11) becomes $\Xi_r \cong \top\wedge\Xi'\wedge\Xi\wedge\top \cong \Xi\wedge\Xi'$ and we're done.

### A.1.8 Bisimulation and Type Equivalence (Lemma 5.2.7)

Inspecting Definition 5.2.6, it is clear that $\Gamma \models P$ is only concerned about transition sequences available from $P$, and not of $P$'s structure (beyond the implicit assumption that $\Gamma \models_{\#} P$ but this is assumed in Lemma 5.2.7 as well). Therefore, having $P \sim P'$, $\Gamma \models P$ if and only if $\Gamma \models P'$. We now focus on the more interesting part of the lemma, that weakening preserves correctness.

Let $\Gamma \models P$, and let $f$ be a strategy function satisfying the requirements of Definition 5.2.6. Let $\Theta \preceq \Gamma$. We show that $\Theta \models P$.

We rely on the fact that $\wr$ commutes with both $\preceq$ and $\searrow$ (if $\Gamma \wr \tilde{\mu} = \Gamma'$ then $\Theta \wr \tilde{\mu} \preceq \Gamma'$, and for any statement $\Gamma'$ with $\Theta' \preceq \Gamma'$, for any projection $\Gamma' \searrow \Gamma''$ there is a projection $\Theta' \searrow \Theta''$ such that $\Theta'' \preceq \Gamma''$. There exists thus a tight matching[1] between the transition network starting from $(\Gamma; P)$ and the

[1]Note that this matching need not be unique because an elementary statement can be weakened to a non-elementary statement, as in $\alpha \succeq \alpha \vee (\beta_1 \wedge \beta_2)$ which has two projections $\alpha \vee \beta_i$. However the proof works no matter which projection is chosen.

one from $(\Theta; P)$, which permits translating $f$ into a strategy function $f'$ for $\Theta \models P$: given a transition sequence from $(\Theta; P)$ to $(\Theta'; P')$, let $(\Gamma'; P')$ be the endpoint of the corresponding sequence from $(\Gamma; P)$. Then $f'(\Theta'; P')$ is the typed process corresponding to $f(\Gamma'; P')$.

Consider an infinite transition sequence as in the Definition but starting with $(\Theta; P) = (\Theta_0; P_0)$. Using the above defined mapping there is a corresponding transition sequence from $(\Gamma; P)$, which, by $\Gamma \models P$, satisfies the requirements of the Definition for some $\alpha$ and $n$, so for all $i$ with $p_i \neq \tau$, $(\alpha \lhd \overline{p_i}_{\mathbf{A}}) \preceq \Gamma'_i$. As $\Gamma'_i \preceq \Theta'_i$, we also have $(\alpha \lhd \overline{p_i}_{\mathbf{A}}) \preceq \Theta'_i$. Secondly, for some $\varepsilon$ with $(\alpha \lhd \varepsilon) \preceq \Gamma_n$ (and therefore $(\alpha \lhd \varepsilon) \preceq \Gamma_n \preceq \Theta_n$), $\alpha \lhd \varepsilon$ is immediately correct for $(\Gamma_n; P_n)$. Inspecting the Definition 6.3.1 for immediate correctness, it only depends on the process type in the third point, and then only for the channel type $\sigma$ of a transition's subject, which is preserved by weakening (weakening may *extend* the channel type mapping but not change or remove a channel's type), so $\alpha \lhd \varepsilon$ must be immediately correct for $(\Theta_n; P_n)$, for the same reason it is immediately correct for $(\Gamma_n; P_n)$.

## A.2 Subject Reduction

In this section we prove Proposition 5.6.3 on the existential type system extended with events and branching resources. This proof is nonetheless also valid for types not containing any $s_{\mathbf{A}}$-resource, i.e. for arbitrary instantiations of the existential type system.

We show that, if $\Gamma \vdash_{\mathcal{K}} P$, $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ then there is $\Gamma_0$ such that $\Gamma_0 \vdash_{\mathcal{K}} P'$ and $\Gamma_0 \preceq \Gamma'$.

Following Lemma A.1.3 we first work on the branchings performed by the transition, and them proceed with the proofs ignoring the (SUM) rule from the LTS.

Let $\Gamma \vdash_{\mathcal{K}} G.P$ and $\hat{\Gamma} \vdash_{\mathcal{K}} G.P + Q$, the latter being obtained from the former using (A-SUM).

From (A-SUM), $\hat{\Gamma} = (\mathsf{sub}(G) + s)_{\mathbf{A}} \lhd \varepsilon \wedge (\Gamma \vee \Gamma_Q)$, for some $\varepsilon$, $s$ and $\Gamma_Q$ depending on $S$. By $\Xi_1 \vee \Xi_2 \succeq \Xi_1$, $\hat{\Gamma} \succeq \left(\sum_{i \in I} p_i\right)_{\mathbf{A}} \lhd \varepsilon \wedge \Gamma$.

Then, (at least) one of the following statements is true:

- $\varepsilon \cong \bot$ (in which case $\hat{\Gamma} \succeq \Gamma$), or
- the transition operator removes it.

We prove this in the beginning of the following subsections as the proof depends on $\mu$.

Secondly, all operators used in the transition operator are either logical homomorphisms or (in the case of process type composition) commute with disjunction. So $(\Gamma_1 \vee \Gamma_2) \wr \mu \cong (\Gamma_1 \wr \mu) \vee (\Gamma_2 \wr \mu)$, and one can assume without loss of generality that the process type being considered contains no disjunction.

We will prove the lemma for $\tau$-reductions, input transitions and output transitions, in that order.

We first consider non-replicated prefixes and then show that if subject reduction holds when consuming non-replicated prefixes, it still holds with replicated prefixes.

## A.2.1 $\tau$-Reductions

First assume $\mu = \tau$. Then, by Lemma A.1.4,

$$P \equiv (\boldsymbol{\nu}\tilde{z})\,(Q \mid \sum_{i\in I} G_i.P_i \mid \sum_{i'\in I'} G_{i'}.P_{i'}), \tag{A.12}$$

and there are $a$, $\hat{\imath} \in I$ and $\hat{\imath}' \in I'$ such that $\mathsf{sub}(G_{\hat{\imath}}) = a$ and $\mathsf{sub}(G_{\hat{\imath}'}) = \bar{a}$.

**Lemma A.2.1 (The Sums are not Active)** *Let $\Gamma \vdash_{\mathcal{K}} P$ where $P$ is given by (A.12).*

*Then $\Gamma$'s local behavioural statement does not contain $(\sum_{i\in I} \mathsf{sub}(G_i))_{\mathbf{A}} \lhd \varepsilon$ or $(\sum_{i'\in I'} \mathsf{sub}(G_{i'}))_{\mathbf{A}} \lhd \varepsilon$ for $\varepsilon \ncong \bot$.*

*Proof* Let

$$(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \vdash_{\mathcal{K}} \sum_{i\in I} G_i.P_i \quad \text{and} \quad (\Sigma'; \Xi'_{\mathrm{L}} \blacktriangleleft \Xi'_{\mathrm{E}}) \vdash_{\mathcal{K}} \sum_{i'\in I'} G_{i'}.P_{i'}$$

with

$$\Xi_{\mathrm{E}} = \bigvee_{j\in J} \Xi_j \quad \text{and} \quad \Xi'_{\mathrm{L}} = \bigvee_{k\in K} \Xi'_k$$

being normal forms of $\Xi_{\mathrm{E}}$ and $\Xi'_{\mathrm{L}}$. Then, assume $\Xi_{\mathrm{L}}$ contains $(\sum_{i\in I} \mathsf{sub}(G_i))_{\mathbf{A}}$ (if it doesn't, we're done). By (A-Sum), $\Xi_{\mathrm{E}}$ must have no concurrent $p_{i'}$:

$$\forall j \in J : \big(\Xi_j \wr \bar{a} \cong \bot \quad \text{or} \quad \forall i \in I \setminus \{\hat{\imath}\} : \Xi_j \wr \overline{\mathsf{sub}(G_i)} \cong \bot\big) \tag{A.13}$$

As $\mathsf{sub}(G_{\hat{\imath}'}) = \bar{a}$, there is $m \neq 0$ s.t. $\bar{a}^m \vdash_{\mathcal{K}} G_{\hat{\imath}'}.P_{\hat{\imath}'}$, which gets carried over by (A-Sum) to $\Xi'_{\mathrm{L}}$ as

$$\bar{a}^m \succeq \Xi'_{\hat{k}} \tag{A.14}$$

for some $\hat{k}$. Now, when applying (E-Par) to type $\sum_{i\in I} G_i.P_i \mid \sum_{i'\in I'} G_{i'}.P_{i'}$, the environment component of the resulting type (see Definition 3.9.1) is:

$$\frac{\bigvee_{j\in J} \Xi_j}{\bigvee_{k\in K} \Xi'_k} = \bigvee_{\rho: K\to J} \bigwedge_{k\in K} \frac{\Xi_{\rho(k)}}{\Xi'_k}$$

Pick an arbitrary $\rho$ and let $j = \rho(\hat{k})$. Then, by (A.13) and (A.14), either $\Xi_j \setminus \Xi'_{\hat{k}} \cong \bot$ (in case $\Xi_j \wr \bar{a} \cong \bot$) or $\Xi_j \setminus \Xi'_{\hat{k}} \preceq \bar{a}^\star \wedge \bigwedge_{i\in I\setminus \hat{\imath}} \overline{\mathsf{sub}(G_i)}^0$ (because $\Xi \wr p = \bot$ iff $p^0 \succeq \Xi$). All $j$ in the first case drop from the disjunction over $\rho$. Using $\Delta \wedge \Delta' \preceq \Delta$, we get $\Xi_{\mathrm{E}} \setminus \Xi'_{\mathrm{L}} \preceq \bar{a}^\star \wedge \bigwedge_{i\in I\setminus\{\hat{\imath}\}} \overline{\mathsf{sub}(G_i)}^0$. In other words, $\big(\sum_{i\in I} \mathsf{sub}(G_i)\big)_{\mathbf{A}}$ is not observable in $(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \odot (\Sigma'; \Xi'_{\mathrm{L}} \blacktriangleleft \Xi'_{\mathrm{E}})$, and is dropped by the application of the clean operator (Definition 6.2.1), as specified in Definition 4.2.6.

It can be similarly shown that $P$'s behavioural statement doesn't contain $(\sum_{i\in I'} \mathsf{sub}(G_i))_{\mathbf{A}} \lhd \varepsilon'$ for $\varepsilon' \ncong \bot$. □

Removing the sums from (A.12) we get

$$(\boldsymbol{\nu}\tilde{z})\,(Q \mid G_{\hat{\imath}}.P_{\hat{\imath}} \mid G_{\hat{\imath}'}.P_{\hat{\imath}'}) \tag{A.15}$$

Lemma A.2.1 implies that (A.12)'s type is stronger than (A.15)'s, so it is now enough to prove subject reduction for transitions not using (Sum).

We can pick $Q = \mathbf{0}$ and $\tilde{z} = \varnothing$, as the general case is an immediate consequence of Lemma A.1.5.

Let $P = \overline{a}\langle\tilde{x}\rangle^{l'}.O \mid a(\tilde{y})^{l}.I$, and consider the transition $P \xrightarrow{\tau} P' = O \mid I\{^{\tilde{x}}/_{\tilde{y}}\}$. We run the typing derivation on both $P$ and $P'$ and show that the former's type is a weakening of the latter's.

Let $\Gamma_O \vdash_{\mathcal{K}} O$ and $\Gamma_I \vdash_{\mathcal{K}} I$. The input's type is $(\boldsymbol{\nu}\tilde{y})\,\Gamma'_I$ where, using (E-Pre),

$$\Gamma'_I = a : \sigma \odot \bigwedge_{k\in\mathcal{K}} \mathsf{prop}_k(\sigma, a(\tilde{y})^{l}, m, m')\odot \\ \overline{\sigma}[\tilde{y}]\lhd \left(\mathsf{dep}_{\mathcal{K}}(a(\tilde{y})^{l}) \wedge (l \vee \bar{a}_{\mathbf{R}})\right) \odot \Gamma_I \lhd \mathsf{dep}_{\mathcal{K}}(a(\tilde{y})^{l}) \tag{A.16}$$

and the output is typed as

$$\Gamma'_O = a : \sigma \odot \bigwedge_{k\in\mathcal{K}} \mathsf{prop}_k(\sigma, \overline{a}\langle\tilde{x}\rangle^{l'}, m, m')\odot \\ \sigma[\tilde{x}]\lhd \left(\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^{l'}) \wedge (l' \vee a_{\mathbf{R}})\right) \odot \Gamma_O \lhd \mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^{l'}). \tag{A.17}$$

Let's first name a few important types and dependencies:

Let $\Gamma = \Gamma'_O \odot (\boldsymbol{\nu}\tilde{y})\,\Gamma'_I$ be the pre-transition type and $\Gamma' = \Gamma_O \odot \Gamma_I\{^{\tilde{x}}/_{\tilde{y}}\}$ the type obtained by re-typing the post-transition process.

We distinguish dependency statements in $\Gamma_I$ for resources based on parameters ($\tilde{y}$) and others, and refer to them using two index sets, respectively $\mathcal{Y}$ and $\mathcal{O}$: the dependency statements in $\Gamma_I$ are

$$\bigwedge_{i\in\mathcal{Y}} \gamma_i \lhd \varepsilon_i \;\wedge\; \bigwedge_{i\in\mathcal{O}} \gamma_i \lhd \varepsilon_i \tag{A.18}$$

with $\forall i \in \mathcal{Y} : \mathrm{n}(\gamma_i) \in \tilde{y}$ and $\forall i \in \mathcal{O} : \mathrm{n}(\gamma_i) \notin \tilde{y}$. We will also need to distinguish between dependencies on parameter resources and other resources, so a dependency $\varepsilon_i$ is sometimes written in the following *normal form* (Lemma 4.2.12):

$$\forall i \in \mathcal{O} \cup \mathcal{Y} : \varepsilon_i = \bigvee_{j\in\mathcal{I}_i} \varepsilon^{O}_{ij} \wedge \varepsilon^{Y}_{ij} \tag{A.19}$$

where $\mathrm{n}(\varepsilon^{O}_{ij}) \cap \tilde{y} = \varnothing$ and $\mathrm{n}(\varepsilon^{Y}_{ij}) \subseteq \tilde{y}$.

We similarly give names to dependencies allowed by the protocol. Just like $\mathcal{Y}$ is an indexing set for resources to be provided by the input side, $\mathcal{Y}'$ is an indexing set for output side resources (as a rule we use a tick $'$ when refering to output-related objects:)

$$\sigma[\tilde{y}] = \bigwedge_{i\in\mathcal{Y}} \gamma_i \lhd \varepsilon^{P}_{i} \quad\text{and}\quad \bar{\sigma}[\tilde{y}] = \bigwedge_{i'\in\mathcal{Y}'} \gamma_{i'} \lhd \varepsilon^{P}_{i'} \tag{A.20}$$

Similarly for $\tilde{x}$ (that may have repeated names unlike $\tilde{y}$):

$$\sigma[\tilde{x}] = \bigwedge_{i\in\mathcal{X}} \gamma_i \lhd \varepsilon^{P}_{i} \quad\text{and}\quad \bar{\sigma}[\tilde{x}] = \bigwedge_{i'\in\mathcal{X}'} \gamma_{i'} \lhd \varepsilon^{P}_{i'} \tag{A.21}$$

We need to subtract one from the local multiplicities from both $a$'s input and output ports, which is permitted by the weakening relation (taken backwards as we're strengthening).

Secondly, existential resources on $a$ and $\bar{a}$ need to be dropped. As we work with non-replicated prefixes, we can assume neither has $\omega$ multiplicity. Moreover they both clearly have a non-zero multiplicity, so that both are either 1 or $\star$. If both are linear then they are no longer observable so those existential resources dropped when applying the erasure operator (Definition 5.1.5). If both are plain then $\mathsf{prop}_k(\sigma, G, \star, \star) = \top$ so there's nothing to prove. If one is plain and the other is linear then only the plain one can have existential resources, but then when composing $\Gamma'_I$ and $\Gamma'_O$ it is no longer observable, so, again, we get that neither $a$ nor $\bar{a}$ have existential resources in $\Gamma'$.

By hypothesis (see beginning of Section 5) we have $\mathbf{R} \in \mathcal{K}$. The elementary responsiveness rule ((5.6) on page 58) gives $\mathsf{prop}_{\mathbf{R}}(\sigma, a(\tilde{y})^l, m, m') = a_{\mathbf{R}} \triangleleft \bar{l} \vee \sigma[\tilde{y}]$ which, composed with (A.20), yields $\mathsf{prop}_{\mathbf{R}}(\sigma, a(\tilde{y})^l, m, m') = a_{\mathbf{R}} \triangleleft (\bar{l} \vee \bigwedge_{i \in \mathcal{Y}} \gamma_i)$.

Applying that transformation in (A.16) and (by strengthening) dropping the dependencies on $\mathsf{dep}_{\mathcal{K}}(a(\tilde{y})^l)$:

$$\Gamma'_I \succeq a : \sigma \odot \bigwedge_{k \in \mathcal{K}} \mathsf{prop}_k(\sigma, a(\tilde{y})^l, m, m') \odot \\ a_{\mathbf{R}} \triangleleft \left( \bar{l} \vee \bigwedge_{i \in \mathcal{Y}} \gamma_i \right) \odot \bar{\sigma}[\tilde{y}] \triangleleft (l \vee \bar{a}_{\mathbf{R}}) \odot \Gamma_I \tag{A.22}$$

The remaining $a_k$ ($k \neq \mathbf{R}$) can be dropped by strengthening (noting that $\alpha \triangleleft \varepsilon \odot \alpha \triangleleft \varepsilon' = \alpha \triangleleft (\varepsilon \wedge \varepsilon') \succeq \alpha \triangleleft \varepsilon'$ if $\alpha$ is universal, and generalises to $\alpha \triangleleft \varepsilon \odot \Xi \succeq \Xi$).

Similarly to (A.18) we use assume the local component of (A.22) has the following normal form:

$$\bigwedge_{i \in \mathcal{Y}} \gamma_i \triangleleft \varepsilon'_i \;\wedge\; \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \varepsilon'_i \;\wedge\; a_{\mathbf{R}} \triangleleft \varepsilon_I. \tag{A.23}$$

The main difference between $\varepsilon_i$ and $\varepsilon'_i$ is due to dependencies getting reduced with $\bar{\sigma}[\tilde{y}]$. Their normal forms is similarly annotated with a tick $'$ :

$$\forall i \in \mathcal{O} \cup \mathcal{Y} : \varepsilon'_i = \bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^O \wedge {\varepsilon'_{ij}}^Y \tag{A.24}$$

where $\mathrm{n}({\varepsilon'_{ij}}^O) \cap \tilde{y} = \varnothing$ and $\mathrm{n}({\varepsilon'_{ij}}^Y) \subseteq \tilde{y}$.

We may now compute $a$'s input responsiveness dependencies $\varepsilon_I$, by reducing $a_{\mathbf{R}} \triangleleft (\bar{l} \vee \bigwedge_{i \in \mathcal{Y}} \gamma_i)$ from (A.22) with statements in (A.24), dropping $\tilde{y}$-based dependencies and any other $a_{\mathbf{R}}$-dependency provided by $\Gamma_I$:

$$\varepsilon_I \succeq \bar{l} \vee \bigwedge_{i \in \mathcal{Y}} \bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^O \tag{A.25}$$

Combining (A.23) and (A.24) we can compute the behavioural statement in $(\boldsymbol{\nu}\tilde{y})\,\Gamma'_I$:

$$(a_{\mathbf{R}} \triangleleft \varepsilon_I) \wedge \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \left( \bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^O \wedge {\varepsilon'_{ij}}^* \right) \tag{A.26}$$

where ${\varepsilon'_{ij}}^*$ is one of $\bot$ (if $p_k \succeq {\varepsilon'_{ij}}^Y$ for some existential $p_k$), $l \vee \bar{a}_{\mathbf{R}}$ (for terms resulting of the composition of ${\varepsilon'_{ij}}^Y$ with the $\bar{\sigma}[\tilde{y}]$-term) or $\top$ (for ${\varepsilon'_{ij}}^Y \cong \top$).

We now proceed to computing Γ's local behavioural statement $\Xi_{\mathrm{L}}$ based on (A.17) and (A.26):

$$\Xi_{\mathrm{L}} \succeq \bigwedge_{k\in\mathcal{K}} \mathsf{prop}_k(\sigma, \overline{a}\langle\tilde{x}\rangle^{l'}, m, m') \odot \sigma[\tilde{x}]\triangleleft \left(\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^{l'}) \wedge (l' \vee a_{\mathbf{R}})\right)\odot$$
$$\Gamma_O \triangleleft \mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^{l'}) \odot a_{\mathbf{R}} \triangleleft \varepsilon_I \wedge \bigwedge_{i\in\mathcal{O}} \gamma_i \triangleleft \bigvee_{j\in\mathcal{I}'_i} \left({\varepsilon'_{ij}}^{O} \wedge {\varepsilon'_{ij}}^{*}\right) \quad \text{(A.27)}$$

Like we did on the $\Gamma_I$-side, dropping the remaining $\bar{a}_k$ and dependencies on $\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^{l'})$, replacing $\bar{a}_{\mathbf{R}}$'s dependencies produced by

$$\mathsf{prop}_k(\sigma, \overline{a}\langle\tilde{x}\rangle^{l'}, m, m') = \bar{l}' \vee \overline{\sigma}[\tilde{x}]$$

when $k = \mathbf{R} \in \mathcal{K}$ by the actual resource set, replacing $a_{\mathbf{R}}$'s dependencies using (A.25), and developing $\sigma[\tilde{x}]$ with (A.20) we get the following (stronger) type:

$$\bar{a}_{\mathbf{R}}\triangleleft \left(\bar{l}' \vee \bigwedge_{i\in\mathcal{X}'} \gamma_i\right) \odot \left(\bigwedge_{i\in\mathcal{X}} \gamma_i \triangleleft \left(\varepsilon_i^P \wedge (l' \vee a_{\mathbf{R}})\right)\right) \odot \Gamma_O \odot$$
$$a_{\mathbf{R}}\triangleleft \left(\bar{l} \vee \bigwedge_{i\in\mathcal{Y}} \bigvee_{j\in\mathcal{I}'_i} {\varepsilon'_{ij}}^{O}\right) \wedge \bigwedge_{i\in\mathcal{O}} \gamma_i \triangleleft \bigvee_{j\in\mathcal{I}'_i} \left({\varepsilon'_{ij}}^{O} \wedge {\varepsilon'_{ij}}^{*}\right)$$

The $a_{\mathbf{R}}$-dependency of the input instantiation term can be reduced with $\varepsilon_I$, the strong dependency replaced by a weak one, and then the $a_{\mathbf{R}}$-term can be dropped, further strengthening the type (replacing the $i$ from $a_{\mathbf{R}}$'s dependencies by $\hat{\imath}$ to avoid name clashes:)

$$\bar{a}_{\mathbf{R}}\triangleleft \left(\bar{l}' \vee \bigwedge_{i\in\mathcal{X}'} \gamma_i\right) \odot \bigwedge_{i\in\mathcal{X}} \gamma_i \triangleleft \left(\varepsilon_i^P \wedge \left(l' \vee \bar{l} \vee \bigwedge_{\hat{\imath}\in\mathcal{Y}} \bigvee_{j\in\mathcal{I}'_i} {\varepsilon'_{\hat{\imath}j}}^{O}\right)\right) \odot$$
$$\Gamma_O \odot \bigwedge_{i\in\mathcal{O}} \gamma_i \triangleleft \left(\bigvee_{j\in\mathcal{I}'_i} {\varepsilon'_{ij}}^{O} \wedge {\varepsilon'_{ij}}^{*}\right)$$

The conjunction on $\hat{\imath} \in \mathcal{Y}$ can be strengthened by keeping only the $\hat{\imath} = i$ factor:

$$\bar{a}_{\mathbf{R}}\triangleleft \left(\bar{l}' \vee \bigwedge_{i\in\mathcal{X}'} \gamma_i\right) \odot \bigwedge_{i\in\mathcal{X}} \gamma_i \triangleleft \left(\varepsilon_i^P \wedge \left(l' \vee \bar{l} \vee \bigvee_{j\in\mathcal{I}'_i} {\varepsilon'_{ij}}^{O}\right)\right) \odot$$
$$\Gamma_O \odot \bigwedge_{i\in\mathcal{O}} \gamma_i \triangleleft \bigvee_{j\in\mathcal{I}'_i} \left({\varepsilon'_{ij}}^{O} \wedge {\varepsilon'_{ij}}^{*}\right)$$

We similarly expand the ${\varepsilon'_{ij}}^{*}$ factors. When $\perp$ they can (by the $\forall\varepsilon : \perp \succeq \varepsilon$ rule) be strengthened to ${\varepsilon'_{ij}}^{Y}\{\tilde{x}/\tilde{y}\}$. Those equal to $\top$ occur precisely when ${\varepsilon'_{ij}}^{Y}\{\tilde{x}/\tilde{y}\} \cong \top$ as well. Finally, ${\varepsilon'_{ij}}^{*} = l \vee \bar{a}_{\mathbf{R}}$ case can be reduced with the

$\bar{a}_{\mathbf{R}} \lhd (\bar{l}' \vee \bigwedge_{i\in\mathcal{X}'} \gamma_i)$-term, resulting in $l \vee (\bar{a}_{\mathbf{R}} \wedge (\bar{l}' \vee \bigwedge_{i\in\mathcal{X}'} \gamma_i))$. That term can be further strengthened into $l \vee \bar{l}' \vee {\varepsilon'_{ij}}^{Y}\{{}^{\tilde{x}}\!/_{\tilde{y}}\}$, resulting in

$$\bigwedge_{i\in\mathcal{X}} \gamma_i \lhd \left(\varepsilon_i^P \wedge \left(l' \vee \bar{l} \vee \bigvee_{j\in\mathcal{I}'_i} {\varepsilon'_{ij}}^{O}\right)\right) \odot \Gamma_O \odot$$

$$\bigwedge_{i\in\mathcal{O}} \gamma_i \lhd \left(\bigvee_{j\in\mathcal{I}'_i} {\varepsilon'_{ij}}^{O} \wedge \left(l \vee \bar{l}' \vee \left({\varepsilon'_{ij}}^{Y}\{{}^{\tilde{x}}\!/_{\tilde{y}}\}\right)\right)\right) \quad \text{(A.28)}$$

We now show that dropping the event annotations from that expression yields an equivalent type, building on the following lemma:

**Lemma A.2.2 (Event Elimination)** *Let $\{\varepsilon_i\}_i$, $\{\varphi_i\}_i$, $\{\varepsilon'_j\}_j$ and $\{\varphi'_j\}_j$ be dependency sets not using the event $l$, and $\{\gamma_i\}_i$, $\{\gamma'_j\}_j$ two resource sets, where $i$ and $j$ are assumed to cover some indexing sets $I$ and $J$. If, for all $i$ and $j$, either $\varepsilon_i \succeq \varepsilon'_j$ or $\varphi_i \preceq \varphi'_j$ holds then $\bigwedge_{i,j} \big(\gamma_i \lhd (\varepsilon_i \wedge (\bar{l} \vee \varphi_i)) \wedge \gamma'_j \lhd ((l \vee \varepsilon'_j) \wedge \varphi'_j)\big) \cong \bigwedge_{i,j} \big(\gamma_i \lhd (\varepsilon_i \wedge \varphi_i) \wedge \gamma'_j \lhd (\varepsilon'_j \wedge \varphi'_j)\big)$.*

We omit the proof but it amounts to showing that, whenever a dependency causes inclusion of any $l \vee \varepsilon'_j$ in a $\bar{l} \vee \varphi_i$ (or vice versa), either dependencies in $\varepsilon'_j$ are also included outside of the $l \vee \ldots$ region, or the entire $l \vee \varepsilon'_j$ becomes $\wedge$-composed with $\bot$, so that the $l \vee \bar{l} \vee \varepsilon \cong \top$ rule becomes redundant, and therefore the events can be omitted.

To remove event annotations from (A.28) we will show that $\forall i' \in \mathcal{X}, i \in \mathcal{O}, j \in \mathcal{I}'_i$, either of the following hold

$$\varepsilon_{i'}^P \succeq {\varepsilon'_{ij}}^{Y} \quad \text{(A.29)}$$

$$\bigvee_{j'\in\mathcal{I}'_{i'}} {\varepsilon'_{i'j'}}^{O} \preceq {\varepsilon'_{ij}}^{O} \quad \text{(A.30)}$$

satisfying the conditions of the Lemma. Specifically, assume that (A.29) does *not* hold. As neither dependency in the inequality use disjunctions, there is $\alpha$ such that (for $\alpha' = \alpha\{{}^{\tilde{x}}\!/_{\tilde{y}}\}$) $\alpha' \preceq {\varepsilon'_{ij}}^{X}$,

$$\alpha \preceq {\varepsilon'_{ij}}^{Y}, \quad \text{(A.31)}$$

$$\alpha \npreceq \varepsilon_{i'}^P. \quad \text{(A.32)}$$

Let $k \in \mathcal{Y}$ be such that $\gamma_k = \alpha$ (see (A.20)). By the definition of parameter instantiation (if $\gamma_{i'}$ does not depend on $\alpha$ then $\alpha$ depends on $\gamma_{i'}$), (A.32) implies

$$\gamma_{i'} \preceq \varepsilon_k^P. \quad \text{(A.33)}$$

As ${\varepsilon'_{ij}}^{Y}$ is taken from $\Gamma'_I$ which is assumed to be closed, we may apply dependency reduction to it and preserve equivalence (i.e. replacing ${\varepsilon'_{ij}}^{Y}$ with the resulting dependency in (A.24) will give a type equivalent to $\Gamma'_I$.)

Inequality (A.31) can also be written ${\varepsilon'_{ij}}^{Y} \cong \alpha \wedge {\varepsilon'_{ij}}^{Y}$. Composing with the $\bar{\sigma}[\tilde{y}] \lhd (l \vee \bar{a}_{\mathbf{R}})$ term from (A.22), or more specifically $\gamma_k \lhd (l \vee (\bar{a}_{\mathbf{R}} \wedge \varepsilon_k^P))$ (remember that $\gamma_k = \alpha$), it becomes $(\alpha * (l \vee \bar{a}_{\mathbf{R}}) \wedge \varepsilon_k^P) \wedge {\varepsilon'_{ij}}^Y$. Applying (A.33) rewritten as $\varepsilon_k^P \cong \varepsilon_k^P \wedge \gamma_{i'}$ we get $(\alpha * (l \vee \bar{a}_{\mathbf{R}}) \wedge \varepsilon_k^P \wedge \gamma_{i'}) \wedge {\varepsilon'_{ij}}^Y$. As $i' \in \mathcal{O}$ we can apply (A.23) and get $(\alpha * (l \vee \bar{a}_{\mathbf{R}}) \wedge \varepsilon_k^P \wedge (\gamma_{i'} * \varepsilon'_{i'})) \wedge {\varepsilon'_{ij}}^Y$ where the meaning of the second $*$ depends on what kind of resource $\gamma_{i'}$ is. Rewriting $\varepsilon'_{i'}$ with (A.24) we get

$$\left(\alpha * (l \vee \bar{a}_{\mathbf{R}}) \wedge \varepsilon_k^P \wedge (\gamma_{i'} * \bigvee_{j' \in \mathcal{I}'_{i'}} {\varepsilon'_{i'j'}}^O \wedge {\varepsilon'_{i'j'}}^Y)\right) \wedge {\varepsilon'_{ij}}^Y \tag{A.34}$$

To summarise, ${\varepsilon'_{ij}}^Y$ can be replaced by (A.34) in $\Gamma'_I$ (A.24), and the resulting type is equivalent, so it can be used instead of $\Gamma'_I$ when computing (A.28).

Dependency (A.34) is strengthened by dropping $l \vee \bar{a}_{\mathbf{R}}$, $\varepsilon_k^P$ and all ${\varepsilon'_{i'j'}}^Y$, and the two $*$ operators are handled like this: if they are conjunctions (for universal resources) then the dependency is strengthened by dropping the resource on their left, otherwise they are left as is and binding replaces the dependency on their left by $\bot$ so in both cases they drop, by $\forall \varepsilon : \bot \vee \varepsilon \cong \varepsilon$. Then, when binding $\tilde{y}$ (A.26), (A.34) becomes $\bigvee_{j' \in \mathcal{I}'_{i'}} {\varepsilon'_{i'j'}}^O \wedge {\varepsilon'_{ij}}^*$. Written in normal form (A.26), we replaced ${\varepsilon'_{ij}}^O$ by ${\varepsilon'_{ij}}^O \wedge \bigvee_{j' \in \mathcal{I}'_{i'}} {\varepsilon'_{i'j'}}^O$, which is equivalent to say that $\bigvee_{j' \in \mathcal{I}'_{i'}} {\varepsilon'_{i'j'}}^O \preceq {\varepsilon'_{ij}}^O$, which is precisely (A.30).

We can therefore apply Lemma A.2.2 to (A.28) twice (for $l$ and then $l'$), getting

$$\Gamma \succeq \bigwedge_{i \in \mathcal{X}} \gamma_i \triangleleft \left(\bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^O \wedge \varepsilon_i^P\right) \odot \Gamma_O \odot \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \left(\bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^O \wedge \left({\varepsilon'_{ij}}^Y \{{}^{\tilde{x}}/_{\tilde{y}}\}\right)\right) \tag{A.35}$$

The factors $\varepsilon_i^P$ can now be strengthened to ${\varepsilon'_{ij}}^Y\{{}^{\tilde{x}}/_{\tilde{y}}\}$ and, comparing with (A.24), observe that the dependencies of $\gamma_i$ for $i \in \mathcal{X}$ and $\mathcal{O}$ are exactly $\varepsilon'_i\{{}^{\tilde{x}}/_{\tilde{y}}\}$:

$$\Gamma \succeq \bigwedge_{i \in \mathcal{X}} (\gamma_i \triangleleft \varepsilon'_i)\,\{{}^{\tilde{x}}/_{\tilde{y}}\} \odot \Gamma_O \odot \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \left(\varepsilon'_i\{{}^{\tilde{x}}/_{\tilde{y}}\}\right) \tag{A.36}$$

As substitution distributes on composition we get $\Gamma \succeq \Gamma'_I\{{}^{\tilde{x}}/_{\tilde{y}}\} \odot \Gamma_O$. In order to reach $\Gamma'$ we still need to transform $\Gamma'_I$ into $\Gamma_I$, i.e. cancel the composition of $\Gamma_I$ with $\bar{\sigma}[\tilde{y}] \triangleleft \bar{a}_{\mathbf{R}}$.

Let $\gamma_i \triangleleft \varepsilon_i \in \Gamma_I$ and consider a resource $\gamma_{i'}$ used in $\varepsilon_i$. Applying the parameter instantiation (A.20) to it replaces it with $\gamma_{i'} * (\bar{a}_{\mathbf{R}} \wedge \varepsilon_{i'}^P)$. If $\gamma_{i'}$ is a universal resource, this can be immediately strengthened back to $\gamma_{i'}$. If it is an existential resource, then the $\bar{a}_{\mathbf{R}} \triangleleft \bar{\sigma}[\tilde{x}]$ term from $\Gamma'_O$ can be applied to $\bar{a}_{\mathbf{R}}$, strengthened to keep only the $\gamma_{i'}$ resource, yielding $\gamma_{i'} \vee (\gamma_{i'} \wedge \varepsilon_{i'}^P)$, which, by factoring $\gamma_{i'}$, is equivalent to $\gamma_{i'} \wedge (\top \vee \varepsilon_{i'}^P)$, itself equivalent to $\gamma_{i'}$. Thus, all dependency reduction due to the output instantiation can be cancelled as long as the output responsiveness term is kept in the type and we get $\Gamma \succeq \Gamma_O \odot \Gamma_I\{{}^{\tilde{x}}/_{\tilde{y}}\} \odot \bar{\sigma}[\tilde{y}] \triangleleft \bar{a}_{\mathbf{R}} \succeq \Gamma_O \odot \Gamma_I\{{}^{\tilde{x}}/_{\tilde{y}}\} = \Gamma'$, as desired.

### A.2.2 Output

Let $\Gamma \vdash_{\mathcal{K}} P \xrightarrow{\overline{a}\langle\tilde{x}\rangle} P'$. Following Lemma A.1.3 we first work on any branching (at most one in this case) performed by the transition, and them proceed with the proofs ignoring the (Sum) rule from the LTS. We already dealt with the disjunction introduced by (E-Sum), and if a branching consumed by the transition is active in $\Gamma$ ($(\sum_i p_i)_{\mathbf{A}}$), then it is removed as specified by Definition 3.7.1. We can now proceed to the sum-less case.

Consider the transition $P = \overline{a}\langle\tilde{x}\rangle^l.Q \xrightarrow{\overline{a}\langle\tilde{x}\rangle} Q$.

Assuming $\Gamma \vdash_{\mathcal{K}} Q$, we get the following type for $P$:

$$\Gamma' = a : \sigma \odot \bigwedge_{k\in\mathcal{K}} \mathsf{prop}_k(\sigma, \overline{a}\langle\tilde{x}\rangle^l, m, m') \odot \\ \sigma[\tilde{x}] \triangleleft \left(\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^l) \wedge (l \vee a_{\mathbf{R}})\right) \odot \Gamma \triangleleft \mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^l) \quad \text{(A.37)}$$

Having $\Gamma'' = \Gamma' \wr \overline{a}\langle\tilde{x}\rangle$, we want to show that $\Gamma \preceq \Gamma''$.

Recall that

$$\Gamma'' \overset{\text{def}}{=} \Gamma' \wr \bar{a} \ \otimes \bar{\sigma}[\tilde{x}] \triangleleft (\bar{a}_{\mathbf{R}} \blacktriangleleft a_{\mathbf{R}}) \quad \text{(A.38)}$$

We first show that multiplicities in $\Gamma''$ are equal or weaker than the ones in $\Gamma$, before proceeding to the dependency statements.

Simplifying (A.37) and (A.38) to only take into account the parts relevant for multiplicities we get $\#\Gamma'' = (\bar{a} \odot \sigma[\tilde{x}] \odot \#\Gamma) \wr \bar{a} \otimes \overline{\sigma}[\tilde{x}]$. By associativity and commutativity of $\odot$ and Lemma A.1.5,

$$\#\Gamma'' \succeq (\bar{a} \odot \#\Gamma) \wr \bar{a} \quad \text{(A.39)}$$

Let $a$'s multiplicities in $\Gamma$ be $\left(a^{m_i} \wedge \bar{a}^{m_o} \blacktriangleleft a^{m'_i} \wedge \bar{a}^{m'_o}\right)$. Then in $\bar{a} \odot \Gamma$ they are

$$\left(a^{m_i} \wedge \bar{a}^{m_o+1} \blacktriangleleft a^{m'_i} \wedge \bar{a}^{m'_o-1}\right)$$

and in $(\bar{a} \odot \Gamma) \wr \bar{a}$ they are

$$\left(a^{m_i} \wedge \bar{a}^{(m_o+1)-1} \blacktriangleleft a^{m'_i-1} \wedge \bar{a}^{m'_o-1}\right).$$

$m'_i - 1 \leq m'_i$, $m'_o - 1 \leq m'_o$ and (by Lemma A.1.5), $(m_o + 1) - 1 \succeq m_o$, so that

$$(\bar{a} \odot \Gamma) \wr \bar{a} \succeq \Gamma \quad \text{(A.40)}$$

Note that $\wr\bar{a}$ and $\backslash\bar{a}$ coincide in this case, as $m_o + 1 \neq \omega$.

Composing (A.39) and (A.40) gives us $\#\Gamma'' \succeq \#\Gamma$, and we now proceed to the dependency statements.

We use the $|\Delta|$ notation to express the set of resources used in a dependency: $|\alpha| = \alpha$ and $|\Delta_1 \wedge \Delta_2| = |\Delta_1 \vee \Delta_2| = |\Delta_1| \cup |\Delta_2|$.

Let $\Omega = |\bar{\sigma}[\tilde{x}]| \setminus \{\bar{a}_k\}_{k\in\mathcal{K}}$, the set of resources to be provided by the output, and $T = (\Omega \cup |\sigma[\tilde{x}]|) \setminus \{\bar{a}_k\}_{k\in\mathcal{K}}$ the set of resources to be provided on one side or the other. In both cases we exclude $\bar{a}_k$ because they interact with the statements introduced by the (E-Pre) rule and have to be handled specially.

We show that each dependency statement in $\Gamma''$ is also present in $\Gamma$, in a possibly weaker form.

Dependencies in $\Gamma''$ are partitioned as follows:

1. $\{\bar{a}_k\}_{k\in\mathcal{K}}$
2. $\Omega$
3. $T \setminus \Omega$
4. $|\Gamma| \setminus (\{\bar{a}_k\}_{k\in\mathcal{K}} \cup T)$

We cover each of those classes in order.
1. $\{\bar{a}_k\}_{k\in\mathcal{K}}$
First consider $k \in \mathcal{E}$. Output $\bar{a}$-existential properties in $\Gamma'$ and $\Gamma''$ may be provided by four different terms. In the following a missing $\bar{a}_k$-statement is written $\bar{a}_k \lhd \bot$.

- $\bar{a}_k \lhd \varepsilon$ as given by $\mathsf{prop}_k(\sigma, \bar{a}\langle\tilde{x}\rangle^l, m, m')$ in the (E-Pre) rule,
- $\bar{a}_k \lhd \varepsilon_c \in \Gamma$,
- $\bar{a}_k \lhd \varepsilon_i \in \sigma[\tilde{x}]$,
- $\bar{a}_k \lhd \varepsilon_o \in \bar{\sigma}[\tilde{x}]$.

In (A.38), the $\Gamma' \wr \bar{a}$ type contains $\bar{a}_k \lhd \bot$, by definition of that operator. The $\otimes\bar{\sigma}[\tilde{x}]$-operation preserves that statement (as it may only weaken existential statements), so $\bar{a}_k$'s dependencies in $\Gamma''$ are equal to those in $\bar{\sigma}[\tilde{x}]$ (after reducing the $\bar{a}_{\mathbf{R}}$-dependency), i.e. $\bar{a}_k \lhd (\varepsilon' \wedge \varepsilon_o) \in \Gamma''$ where $\varepsilon'$ is $\bar{a}_{\mathbf{R}}$'s dependencies in $\Gamma''$.

First assume $\varepsilon_o \cong \bot$. Then $\bar{a}_k \lhd \bot \in \Gamma''$. The $\forall\varepsilon : \varepsilon \preceq \bot$ rule gives $\varepsilon_c \preceq \bot$ as required.

Now assume that $\varepsilon_o \ncong \bot$ but $\varepsilon \cong \bot$. The first case implies that $\bar{a}_k \in |\bar{\sigma}[\tilde{x}]|$, i.e. $\bar{a}_{\mathbf{R}} \lhd \bar{a}_k \preceq \Gamma'$, which reduces with $\bar{a}_k \lhd \bot$ to give $\bar{a}_{\mathbf{R}} \lhd \bot$, i.e. $\varepsilon' \cong \bot$, which itself causes $\bar{a}_k \lhd \bot \in \Gamma''$, and $\varepsilon_c \preceq \bot$ concludes the case once more.

Now assume both $\varepsilon_o \ncong \bot$ and $\varepsilon \ncong \bot$. Since $\Gamma' \wr \bar{a}$ is well-defined, $m'_i > 0$. Since $\bar{a}_k \in |\bar{\sigma}[\tilde{x}]|$, Convention 5.0.4 applies to forbid the type to have blocked liveness, i.e. there is $\bar{a}^m \in \sigma[\tilde{x}]$ with $m > 0$. Because (E-Pre) introduces that type into $\Gamma'$, $m_i > 0$ as well. The sum of two non-zero multiplicities being $\star$,[2] the side condition in $\mathsf{prop}_{\mathbf{A}}$ requires $m_o + m'_o \notin \{1; \star\}$. Since $\Gamma'$ includes $\bar{a}^1$, $m_o > 1$. This excludes $m_o + m'_o = 0$, leaving only $m_o + m'_o = \omega$. Therefore this only holds in the second form of the structural lemma.

Now assume $k \in \mathcal{U}$, and let $\bar{a}_k \lhd \varepsilon_0 \in \Gamma$. Then $\exists\varepsilon' : \bar{a}_k \lhd \varepsilon' \in \Gamma'$ and $\varepsilon' \succeq \varepsilon_0$. Then let $\bar{a}_k \lhd \varepsilon'' \in \Gamma''$. We have $\varepsilon'' \succeq \varepsilon'$ so that $\varepsilon'' \succeq \varepsilon_0$, as required.
2. $\Omega$
We first calculate $\bar{a}_{\mathbf{R}}$'s dependencies in $\Gamma'$.

Having $\forall\alpha \in \Omega : \alpha \lhd \varepsilon_\alpha \in \Gamma$, fix a set of $\varepsilon_{\alpha i}$ and $\varepsilon'_{\alpha i}$ and indexing sets $I_\alpha$ such that:

$$\varepsilon_\alpha \cong \bigvee_{i \in I_\alpha} (\varepsilon_{\alpha i} \wedge \varepsilon'_{\alpha i}) \tag{A.41}$$

and $|\varepsilon_{\alpha i}| \cap \Omega = \varnothing$, $|\varepsilon'_{\alpha i}| \subseteq \Omega$ for all $\alpha$ and $i$.

[2]This is a crucial requirement of the proof — if there were two non-zero multiplicities $m_1$ and $m_2$ such that $m_1 + m_2 \neq \star$ then a channel type with $1^{m_1}$ and $\bar{1}_k$ in $\xi_O$ and $1^{m_2}$ in $\xi_I$ would not have blocked liveness but subject reduction would not hold for transitions like $\bar{a}\langle a\rangle \xrightarrow{\bar{a}\langle a\rangle} \mathbf{0}$.

Let $\Phi$ be the set of functions $\varphi$ such that $\mathrm{dom}(\varphi) = \Omega$ and $\forall \alpha \in \Omega$, $\varphi(\alpha) \in I_\alpha$. We say that such a function is *at a circularity* if $\sigma[\tilde{x}] \odot \bigwedge_{\alpha \in \Omega}(\varepsilon_{\alpha\varphi(\alpha)})$ contains $\alpha \triangleleft \bot$ for some $\alpha \in \Omega$.

Define $\varepsilon_\varphi$ to be $\bot$ if $\varphi(\Omega)$ is at a circularity, $\top$ otherwise. Having $\bar{a}_{\mathbf{R}} \triangleleft \varepsilon_0 \in \Gamma$ (or $\varepsilon_0 = \top$ if there is no such statement), $\bar{a}_{\mathbf{R}} \triangleleft \varepsilon' \in \Gamma'$, with:

$$\varepsilon' \cong \varepsilon_0 \wedge \bigvee_{\varphi \in \Phi} \left( \varepsilon_\varphi \wedge \bigwedge_{\alpha \in \Omega} \varepsilon'_{\alpha\varphi(\alpha)} \right) \tag{A.42}$$

Finally, having $\bar{\sigma}[\tilde{x}] = \left(\tilde{x} : \tilde{\sigma}; \tilde{u}_L \wedge \tilde{\delta}_L \blacktriangleleft \tilde{u}_E \wedge \tilde{\delta}_E\right)$, let $\forall \alpha \in \Omega : \alpha \triangleleft \varepsilon_{\alpha 0} \in \tilde{\delta}_L$. Then, $\forall \alpha \in \Omega : \alpha \triangleleft \varepsilon''_\alpha \in \Gamma''$, with

$$\varepsilon''_\alpha \succeq \bigvee_{\varphi \in \Phi} \left( \varepsilon_{\alpha 0} \ \wedge \ \varepsilon_\varphi \ \wedge \ \bigwedge_{\alpha' \in \Omega} \varepsilon'_{\alpha'\varphi(\alpha')} \right) \tag{A.43}$$

That equation gives a stronger form of $\varepsilon''_\alpha$ where we removed $\varepsilon_0$ from (A.42) as well as statements for resources $\alpha$ contained in $|\sigma[\tilde{x}]| \cap |\bar{\sigma}[\tilde{x}]|$, produced by $\sigma[\tilde{x}]$ in $\Gamma'$. Note that those statements may only be statements on universal resources by Convention 5.0.4, and will therefore be added to $\varepsilon_\alpha$ using the $\wedge$ operator, so that they may be dropped by applying the $\forall \varepsilon_1 \varepsilon_2 : \varepsilon_1 \wedge \varepsilon_2 \succeq \varepsilon_1$ rule.

The following lemma says that if $\bar{a}_{\mathbf{R}}$'s dependencies isn't $\bot$ then all dependencies of local resources on remote resources in $\Gamma$ are contained in the protocol (to be precise, by parameter instantiation of local resources, which includes dependencies added to complete the protocol).

**Lemma A.2.3 (Protocol Satisfaction)** *Let $\Omega$, $\Phi$, $\varepsilon_{\alpha i}$ and $\varepsilon'_{\alpha i}$ be defined as before (for all $\alpha \in \Omega$ and $i \in \{0\} \cup I_\alpha$), and $\varphi \in \Phi$ be a function that is* not *at a circularity.*

*Then, for all $\alpha$, $\varepsilon_{\alpha\varphi(\alpha)} \preceq \varepsilon_{\alpha 0}$.*

*Proof* $\varepsilon_{\alpha\varphi(\alpha)} \cong \tilde{\beta}_s \wedge \tilde{\beta}_w$ with $\tilde{\beta}_s \subseteq \tilde{\beta}_w$ and similarly let $\varepsilon_{\alpha 0} \cong (\tilde{\beta}'_s <) \wedge (\tilde{\beta}'_w \leq)$ with $\tilde{\beta}'_s \subseteq \tilde{\beta}'_w$.

We show by contradiction that

$$\tilde{\beta}_s \subseteq \tilde{\beta}'_s \quad \wedge \quad \tilde{\beta}_w \subseteq \tilde{\beta}'_w. \tag{A.44}$$

Let $\beta \in \tilde{\beta}_s \setminus \tilde{\beta}'_s$. Because $\beta \notin \tilde{\beta}'_s$, $\beta \triangleleft \alpha \preceq \sigma[\tilde{x}]$, which, when composed with $\alpha \triangleleft \varepsilon_{\alpha\varphi(\alpha)}$, yields $\alpha \triangleleft \bot$, contradicting $\varphi$ not being at a circularity.

Now let $\beta \in \tilde{\beta}_w \setminus \tilde{\beta}'_w$. Similarly to the other case we obtain that $\beta \triangleleft \alpha \preceq \sigma[\tilde{x}]$, which, again produces $\alpha \triangleleft \bot$, a contradiction.

Applying $\forall \varepsilon_1, \varepsilon_2 : \varepsilon_1 \preceq \varepsilon_1 \wedge \varepsilon_2$ on (A.44) yields $(\tilde{\beta}_s <) \preceq (\tilde{\beta}'_s <)$ and $(\tilde{\beta}_w \leq) \preceq (\tilde{\beta}'_w \leq)$, and therefore $\varepsilon_{\alpha\varphi(\alpha)} \preceq \varepsilon_{\alpha 0}$. □

We claim that (A.43) is weaker than $\alpha \triangleleft \varepsilon_\alpha$ which is in $\Gamma$:

First, for all $\varphi \in \Phi$ and $\alpha \in \Omega$, taking $i = \varphi(\alpha)$, $\varepsilon_{\alpha i} \preceq \varepsilon_{\alpha 0} \wedge \varepsilon_\varphi$: If $\varphi$ is at a circularity then the inequality is an immediate consequence of $\forall \varepsilon : \varepsilon \preceq \bot$. if $\varphi$ is not at a circularity then $\varepsilon_\varphi = \top$ and the inequality is proved in Lemma A.2.3.

Second, $\bigwedge_{\alpha' \in \Omega} \varepsilon_{\alpha'\varphi(\alpha')} \preceq \varepsilon_{\alpha i}$, as a direct application of the $\varepsilon_1 \wedge \varepsilon_2 \preceq \varepsilon_1$ rule (as $\varphi(\alpha) = i$).

3. $T \setminus (\{\bar{a}_k\}_{k \in \mathcal{K}} \cup \Omega)$

Let $\alpha \in T$ (and not in $\{\bar{a}_k\}_{k\in\mathcal{K}} \cup \Omega$), with $\alpha \triangleleft \varepsilon_\alpha \in \Gamma$, $\varepsilon'_\alpha \in \sigma[\tilde{x}]$. Then $\varepsilon_\alpha \wedge \varepsilon'_\alpha \in \Gamma'$ with the additional dependency $(\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle) \wedge a_{\mathbf{R}})$ if $l \notin \tilde{l}$. Then, by definition of the "$\otimes(\overline{\sigma}[\tilde{x}] \triangleleft (\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^{l}) \wedge (l \vee \bar{a}_{\mathbf{AR}})))$" operation, $\exists\alpha'' \triangleleft \varepsilon$ s.t. $\varepsilon''_\alpha \in \Gamma''$ and $\varepsilon''_\alpha \preceq \varepsilon_\alpha$.

4. $|\Gamma| \setminus (\{\bar{a}_k\}_{k\in\mathcal{K}} \cup T)$

Those resources have, in both $\Gamma'$ and $\Gamma''$ and compared to $\Gamma$, just the additional dependency $\mathsf{dep}_{\mathcal{K}}(\overline{a}\langle\tilde{x}\rangle^{l})$ which can be removed by strengthening.

### A.2.3 Input

Let $(\Gamma'; P') \xrightarrow{a(\tilde{x})} (\Gamma''; P'')$, where $P' = a(\tilde{y})^{l}.P$, $\Gamma \vdash_{\mathcal{K}} P$ and $\Gamma' \vdash_{\mathcal{K}} P'$.

By the Renaming Lemma, $\Gamma\{^{\tilde{x}}/_{\tilde{y}}\} \vdash_{\mathcal{K}} P\{^{\tilde{x}}/_{\tilde{y}}\}$.

### A.2.4 Replication

Let $P \equiv (!\,G).P_0 \mid Q$ and consider a transition $P \xrightarrow{\mu} P' \equiv P \mid P_0\{^{\tilde{x}}/_{\tilde{y}}\}$ (so $\mu \neq \tau$, but the transformation given below can straightforwardly be extended to transitions involving two guarded prefixes, for the $\mu = \tau$-case). For readability purposes we omitted a restriction "$\boldsymbol{\nu}\tilde{a}$" before $P$ that would be needed for full generality, but the proof is the same.

Let $\mathsf{sub}(G) = \mathsf{sub}(\mu) = p$, $\mathsf{obj}(G) = \tilde{y}$ and $\mathsf{obj}(\mu) = \tilde{x}$ (they may be different in case $G$ is an input), and set, for all $k \in \mathcal{K}$, $\Xi_k = \mathsf{prop}_k(\sigma, !\,G, m+\omega, m')$.

Following the type system rule (E-Pre), $P$'s type $\Gamma$ is as follows:

$$\Gamma = \Big(p : \sigma; p^{\omega} \blacktriangleleft p^{m} \wedge \bar{p}^{m'}\Big) \odot !\,(\boldsymbol{\nu}\tilde{z})\,\big( \bigwedge_{k\in\mathcal{K}} \Xi_k \odot \\ \overline{\sigma}[\tilde{y}] \triangleleft (\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}}) \odot \Gamma_0 \triangleleft \mathsf{dep}_{\mathcal{K}}(G)\big) \odot \Gamma_Q \quad \text{(A.45)}$$

where $\Gamma_0 \vdash_{\mathcal{K}} P_0$ and $\Gamma_Q \vdash_{\mathcal{K}} Q$.

The proof involves extracting one element of the replicated process (as if we invoked the usual rule $!\,P \mapsto (P \,|\, !\,P)$, which, remember, is not part of out notion of structural congruence because a port with multiplicity $\omega$ should not appear more than once in a process).

Let $\hat{P} = \hat{G}.P_0 \mid (!\,G).P_0 \mid Q$ where $\hat{G}$ is $G$ but with $\mathsf{sub}(\hat{G}) = q$ instead of $p$, for some fresh port $q$ (input if $p$ is an input and output if $p$ is an output). Note that we keep $\mathsf{obj}(\hat{G}) = \mathsf{obj}(G)$ so if for instance $G = \overline{a}\langle a\rangle$ then we set $\hat{G} = \overline{b}\langle a\rangle$ for some fresh $b$, *not* "$\overline{b}\langle b\rangle$".

Observe that $\hat{P} \xrightarrow{\hat{\mu}} P'$ (where, again, $\hat{\mu}$ is such that $\hat{\mu}\{^{\mathsf{sub}(G)}/_{t}\} = \mu$ and $\mathsf{obj}(\hat{\mu}) = \mathsf{obj}(\mu)$). Similarly to (A.45), $\hat{P}$ has type $\hat{\Gamma}$, in which $\forall k \in \mathcal{K} : \hat{\Xi}_k = \mathsf{prop}_k(\sigma, \hat{G}, \star, \star)$:

$$\hat{\Gamma} = \big(q : \sigma; q^{1} \blacktriangleleft\big) \odot (\boldsymbol{\nu}\tilde{z})\,\big( \bigwedge_{k\in\mathcal{K}} \hat{\Xi}_k \odot \overline{\sigma}[\tilde{y}] \triangleleft (\mathsf{dep}_{\mathcal{K}}(\hat{G}) \wedge \bar{q}_{\mathbf{R}}) \odot \Gamma_0 \triangleleft \mathsf{dep}_{\mathcal{K}}(\hat{G})\big) \odot \\ \Big(p : \sigma; p^{\omega} \blacktriangleleft p^{m} \wedge \bar{p}^{m'-1}\Big) \odot !\,(\boldsymbol{\nu}\tilde{z})\,\big( \bigwedge_{k\in\mathcal{K}} \Xi_k \odot \\ \overline{\sigma}[\tilde{y}] \triangleleft (\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}}) \odot \Gamma_0 \triangleleft \mathsf{dep}_{\mathcal{K}}(G)\big) \odot \Gamma_Q \quad \text{(A.46)}$$

Observe that we set $\bar{p}$'s remote multiplicities to $m'-1$ rather than just $m'$ like in (A.45), as our goal is to have $\Gamma \wr \mu$ and $\hat{\Gamma} \wr \hat{\mu}$ be as close as possible so that subject reduction with non-replicated guards on the latter can be used to describe the former. We still have $\hat{\Gamma} \vdash_{\mathcal{K}} \hat{P}$ as (E-PRE) doesn't put any restriction on remote multiplicities of the complement.

Define a set of $\Gamma_i$ and $\hat{\Gamma}_i$ such that:

$$\bigwedge_{k\in\mathcal{K}\setminus\{\mathbf{R}\}} \Xi_k \odot \overline{\sigma}[\tilde{y}]\triangleleft(\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}}) \odot \Gamma_0\triangleleft \mathsf{dep}_{\mathcal{K}}(G) = \bigvee_{i\in I}\Gamma_i \tag{A.47}$$

$$\bigwedge_{k\in\mathcal{K}\setminus\{\mathbf{R}\}} \hat{\Xi}_k \odot \overline{\sigma}[\tilde{y}]\triangleleft(\mathsf{dep}_{\mathcal{K}}(\hat{G}) \wedge \bar{q}_{\mathbf{R}}) \odot \Gamma_0\triangleleft \mathsf{dep}_{\mathcal{K}}(\hat{G}) = \bigvee_{i\in I}\hat{\Gamma}_i \tag{A.48}$$

As $q$ is fresh, $\mathrm{n}(q)$ doesn't appear in $\Gamma_0$ or $\overline{\sigma}[\tilde{y}]$ and

$$\forall i \in I : \hat{\Gamma}_i\{^{\mathrm{n}(p)}/_{\mathrm{n}(q)}\} = \Gamma_i \tag{A.49}$$

By definition of $\mathsf{prop}_{\mathbf{R}}$, $\hat{\Xi}_{\mathbf{R}} = p_{\mathbf{R}}\triangleleft\sigma[\tilde{y}]$. As $\odot$ and $\boldsymbol{\nu}\tilde{z}$ are logical homomorphisms, the $q$-part from (A.46) under $\tilde{z}$-replication is:

$$\begin{aligned}(\boldsymbol{\nu}\tilde{z})\,\big(q_{\mathbf{R}}\triangleleft\sigma[\tilde{y}] \odot \bigwedge_{k\in\mathcal{K}\setminus\{\mathbf{R}\}} \hat{\Xi}_k \odot \overline{\sigma}[\tilde{y}]\triangleleft(\mathsf{dep}_{\mathcal{K}}(\hat{G}) \wedge \bar{q}_{\mathbf{R}}) \odot \Gamma_0\triangleleft \mathsf{dep}_{\mathcal{K}}(\hat{G})\big)\\
&= (\boldsymbol{\nu}\tilde{z})\bigvee_{i\in I}\big(q_{\mathbf{R}}\triangleleft\sigma[\tilde{y}] \odot \hat{\Gamma}_i\big)\\
&= \bigvee_{i\in I}(\boldsymbol{\nu}\tilde{z})\,\big(q_{\mathbf{R}}\triangleleft\sigma[\tilde{y}] \odot \hat{\Gamma}_i\big)\\
&\cong \bigvee_{i\in I}\big(q_{\mathbf{R}}\triangleleft\hat{\varepsilon}_i \wedge (\boldsymbol{\nu}\tilde{z})\,\hat{\Gamma}_i\big)\end{aligned} \tag{A.50}$$

for some collection of $\hat{\varepsilon}_i$ (which are $\sigma[\tilde{y}]$ "transformed" according to $\hat{\Gamma}_i$ by the reduction itself performed by $\odot$). Similarly,

$$(\boldsymbol{\nu}\tilde{z})\,\big(p_{\mathbf{R}}\triangleleft\sigma[\tilde{y}] \odot \bigwedge_{k\in\mathcal{K}\setminus\{\mathbf{R}\}} \Xi_k \odot \overline{\sigma}[\tilde{y}]\triangleleft(\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}}) \odot \Gamma_0\triangleleft \mathsf{dep}_{\mathcal{K}}(G)\big) \cong \bigvee_{i\in I}\big(p_{\mathbf{R}}\triangleleft\varepsilon_i \wedge (\boldsymbol{\nu}\tilde{z})\,\Gamma_i\big)$$

for some set of $\varepsilon_i$. Replicating that type gives:

$$\mathbf{!}\,(\boldsymbol{\nu}\tilde{z})\,\big(\bigwedge_{k\in\mathcal{K}} \Xi_k \odot \overline{\sigma}[\tilde{y}]\triangleleft(\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}}) \odot \Gamma_0\triangleleft \mathsf{dep}_{\mathcal{K}}(G)\big) \cong \bigvee_{J\subseteq I}\bigodot_{j\in J}\big(p_{\mathbf{R}}\triangleleft\varepsilon_j \odot (\boldsymbol{\nu}\tilde{z})\,\Gamma_j^2\big) \tag{A.51}$$

We are now ready to compute $\Gamma \wr \mu$ and $\hat{\Gamma} \wr \hat{\mu}$. Since the definition of $\wr$ (Definition 5.1.6, page 50) is slightly different for inputs and outputs in the polarity of the composition operator ($\odot$ for inputs and $\otimes$ for outputs) and of the parameter instantiation ($\sigma[\tilde{x}]$ for inputs and $\overline{\sigma}[\tilde{x}]$ for outputs) we now assume $\mu$ is an output. The proof for inputs is identical, with the two above changes applied everywhere.

Using $(\Gamma \odot \Gamma') \wr p \;\succeq\; (\Gamma \wr p) \odot \Gamma'$ and (A.51):

$$\Gamma \wr \mu \succeq \left(\left(p : \sigma; p^\omega \blacktriangleleft p^m \wedge \bar{p}^{m'-1}\right) \odot \bigvee_{J\subseteq I} \bigodot_{j\in J} \left(p_{\mathbf{R}} \triangleleft \varepsilon_j \wedge (\boldsymbol{\nu}\tilde{z})\, \Gamma^2_j\right)\right) \otimes \overline{\sigma}[\tilde{x}] \triangleleft p_{\mathbf{R}} \tag{A.52}$$

Noting that $\vee$ is idempotent (so counting one item more than once is not a problem) we have the following equality:

$$\bigvee_{J\subseteq I} \Delta_J \cong \bigvee_{i\in I} \bigvee_{\substack{J\subseteq I\\ J\ni i}} \Delta_J$$

Moreover, $p_{\mathbf{R}} \triangleleft \varepsilon_1 \odot p_{\mathbf{R}} \triangleleft \varepsilon_2 \succeq p_{\mathbf{R}} \triangleleft \varepsilon_1$, so, in (A.52), we may move $p_{\mathbf{R}} \triangleleft \varepsilon_j$ outside the composition:

$$\Gamma \wr \mu \succeq \left(\left(p : \sigma; p^\omega \blacktriangleleft p^m \wedge \bar{p}^{m'-1}\right) \odot \bigvee_{i\in I} \left(p_{\mathbf{R}} \triangleleft \varepsilon_i \wedge \bigvee_{\substack{J\subseteq I\\ J\ni i}} \bigodot_{j\in J} (\boldsymbol{\nu}\tilde{z})\, \Gamma^2_j\right)\right) \otimes \overline{\sigma}[\tilde{x}] \triangleleft p_{\mathbf{R}} \tag{A.53}$$

Moving on to $\hat{\Gamma}$ and $\hat{\Gamma} \wr \hat{\mu}$:

$$\begin{aligned}\hat{\Gamma} \wr \hat{\mu} \preceq \Bigg(&\left(p : \sigma; p^\omega \blacktriangleleft p^m \wedge \bar{p}^{m'-1}\right) \odot \bigvee_{J\subseteq I} \bigodot_{j\in J} \left(p_{\mathbf{R}} \triangleleft \varepsilon_j \wedge (\boldsymbol{\nu}\tilde{z})\, \Gamma^2_j\right)\\ &\odot \left(q : \sigma; q^0 \blacktriangleleft\ \right) \odot \bigvee_{i\in I} \left(q_{\mathbf{R}} \triangleleft \hat{\varepsilon}_i \wedge (\boldsymbol{\nu}\tilde{z})\, \hat{\Gamma}_i\right)\Bigg) \otimes \overline{\sigma}[\tilde{x}] \triangleleft q_{\mathbf{R}}\end{aligned} \tag{A.54}$$

The $\left(q : \sigma; q^0 \blacktriangleleft\ \right)$ factor is neutral for $\odot$ (it is $\cong$-equivalent to $\top$) so we may drop it. We have $\forall i : \hat{\varepsilon}_i \succeq \varepsilon_i$ as the latter may have "captured" responsiveness of additional $p$-prefixes found in $\Gamma_0$ (the continuation). As $\triangleleft$ is contravariant on the right with respect to $\preceq$ (Definition 3.6.1, page 22), $\forall i : q_{\mathbf{R}} \triangleleft \hat{\varepsilon}_i \preceq q_{\mathbf{R}} \triangleleft \varepsilon_i$, so (A.54) becomes

$$\begin{aligned}\hat{\Gamma} \wr \hat{\mu} \preceq \Bigg(&\left(p : \sigma; p^\omega \blacktriangleleft p^m \wedge \bar{p}^{m'-1}\right) \odot \bigvee_{J\subseteq I} \bigodot_{j\in J} \left(p_{\mathbf{R}} \triangleleft \varepsilon_j \wedge (\boldsymbol{\nu}\tilde{z})\, \Gamma^2_j\right)\\ &\odot \bigvee_{i\in I} \left(q_{\mathbf{R}} \triangleleft \varepsilon_i \wedge (\boldsymbol{\nu}\tilde{z})\, \hat{\Gamma}_i\right)\Bigg) \otimes \overline{\sigma}[\tilde{x}] \triangleleft q_{\mathbf{R}}\end{aligned} \tag{A.55}$$

Let's call that type $\Gamma_M$ ("$M$" as it is in some sense "in the Middle" between $\Gamma \wr \mu$ and $\hat{\Gamma} \wr \hat{\mu}$). Inequality (A.55) can then be written $\Gamma_M \succeq \hat{\Gamma} \wr \hat{\mu}$, or

$$\Gamma_M\{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \succeq \hat{\Gamma} \wr \hat{\mu}\{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \tag{A.56}$$

Applying (A.49) and the definition of replication ($\Gamma \odot \mathbf{!}\,\Gamma = \mathbf{!}\,\Gamma$), (A.53) becomes

$$\Gamma \wr \mu \succeq \Gamma_M\{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \tag{A.57}$$

We have already shown subject reduction for transitions using non-replicated guards, so there is $\Gamma'$ such that $\hat{\Gamma}' \wr \hat{\mu} \succeq \Gamma'$ and $\Gamma' \vdash_{\mathcal{K}} P'$. The first equation implies

$$\hat{\Gamma}' \wr \hat{\mu}\{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \succeq \Gamma'\{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \tag{A.58}$$

As $\mathrm{n}(q)$ doesn't appear in $P'$, it doesn't appear in $\Gamma'$ either so

$$\Gamma'\{^{\mathrm{n}(p)}/_{\mathrm{n}(q)}\} = \Gamma' \tag{A.59}$$

Chaining (A.57), (A.56), (A.58) and (A.59) we get $\Gamma \wr \mu \succeq \Gamma'$, as required.

## A.3 Simple Correctness

As a pre-requisite to soundness we show the following lemma:

**Lemma A.3.1 (Simple Correctness)** *Let* $\Gamma \vdash P$. *Then* $\Gamma \models_{\#} P$.

The following auxiliary lemma says that operators used by the type system may only increase or preserve local multiplicities:

**Lemma A.3.2** *Let* $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *and* $\Gamma'$ *be process types and let* $p^m \succeq \Xi_{\mathrm{L}}$.

- *If* $\Gamma \odot \Gamma'$ *is well defined and equal to some* $(\Sigma'; \Xi'_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *then* $\exists m' \geq m$ *s.t.* $p^{m'} \succeq \Xi'_{\mathrm{L}}$.
- *If* $(\boldsymbol{\nu} a)\,\Gamma$ *is equal to some* $(\Sigma'; \Xi'_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$, *with* $a \neq \mathrm{n}(p)$ *then* $p^m \succeq \Xi'_{\mathrm{L}}$.

We omit the proof, which is an easy consequence of properties of the $+$ operator on multiplicities.

The following lemma is used when proving that the type system guarantees uniformity of $\omega$-names:

**Lemma A.3.3** *Let* $(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \vdash P$ *with* $p^\omega \succeq \Xi_{\mathrm{L}}$. *Then* $p$ *appears at most once in* $P$ *in subject position, and, in case that occurs,* $P = C[\,!\,T.Q]$ *where* $T$*'s subject is* $p$ *and* $C$ *doesn't bind* $\mathrm{n}(p)$.

*Proof* The type system performs the following operations on local multiplicities:

1. Prefix rules add 1 or $\omega$ for prefix subjects
2. Prefix rules $\odot$-compose the remote behaviour (for objects)

Therefore $p^\omega$ is produced by the type system whenever $p$ is the subject of a replicated prefix $!\,(\boldsymbol{\nu}\tilde{z})\,a(\tilde{y}).P'$ or $!\,(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle.P'$ (point 1), and when it is a free parameter of an output (point 2, with the appropriate $\omega$ multiplicity in the channel type).

More than one occurence of a port would result in composition of two non-zero multiplicities and give $p^\star$, so exactly one of the above cases must occur.

Then, a local $p^\omega$ channel usage is preserved by composition (only with types having $p^0$ on the local side), prefixing (only with ports other than $p$) and binding (only of names other than $\mathrm{n}(p)$). $\square$

We work on a restricted form of simple correctness that does not permit arbitrary transition sequences:

**Definition A.3.4 (Simple Correctness Predicate)** *A typed process* $(\Gamma; P)$ *is said* locally correct *with respect to simple semantics (written* $\mathsf{good}_{\#}(\Gamma; P)$*) if it satisfies Definition 3.12.1 whenever* $\tilde{\mu} = \varnothing$.

Then this lemma, together with Subject Reduction, will be used for full generality:

**Lemma A.3.5 (Weakening Preserves Local Correctness)**
*Let* $(\Gamma; P)$ *be a typed process with* $\mathsf{good}_{\#}(\Gamma; P)$*, and* $\Gamma' \succeq \Gamma$*. Then* $\mathsf{good}_{\#}(\Gamma'; P')$ *also holds*

**Lemma A.3.6 (Local Correctness Lemma)** *If* $\Gamma \vdash P$ *then* $\mathsf{good}_{\#}(\Gamma; P)$.

*Proof*
Let $\Gamma \vdash P$, with $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$. The items below corresponds to those in Definition 3.12.1. We show for the case where $\tilde{\mu} = \varnothing$, which is generalised for any transition sequence using subject reduction.

1. is easily shown by induction on the length of the typing derivation.

2. Let $P \xrightarrow{\mu} P'$. We distinguish the cases $\mu = \tau$ and $\mu \neq \tau$:

   (a) $\mathsf{sub}(\mu) = p$. By Lemma A.1.4, $P \equiv P' = (\boldsymbol{\nu}\tilde{z})\,(G.Q \mid R)$ (modulo replication, and with $\mathrm{n}(p) \notin \tilde{z}$). By subject congruence, $\Gamma \vdash P'$.
   Let $\Gamma_Q \vdash Q$, $\Gamma_R \vdash R$ and $\mathsf{obj}(G) = \tilde{x}$. Then, typing $P'$ uses (R-Pre), (R-Par) and (R-Res), resulting in

$$\begin{aligned}\Gamma = (\boldsymbol{\nu}\tilde{z})\Big[\Big(p : \sigma;\ \blacktriangleleft p^{m_0} \wedge \bar{p}^{m_0'}\Big) \odot \Big(;p^{\#(G)} \blacktriangleleft\Big) \odot \\ !_{\text{if } \#(G) = \omega}\ (\boldsymbol{\nu}\mathrm{bn}(G))\Big(\ \Gamma \triangleleft \mathsf{dep}_{\mathcal{K}}(G) \odot \overline{\sigma}[\tilde{x}] \triangleleft (\mathsf{dep}_{\mathcal{K}}(G) \wedge (l \vee \bar{p}_{\mathbf{R}})) \odot \\ \Big(;\bigwedge_{k \in \mathcal{K}} \mathsf{prop}_k(\sigma, G, m, m') \blacktriangleleft\Big)\Big)\Big] \odot \Gamma_R \qquad \text{(A.60)}\end{aligned}$$

   In (A.60), $m_0$ must be equal to $\#(G)$ or $\star$ in order for the first composition to be well-defined, so, by Lemma A.3.2, $p^m \succeq \Xi_{\mathrm{L}}$ for some $m \neq 0$.
   Let $\Gamma_+ = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}} \odot \bar{p}^{\star})$, making $\Gamma_+ \wr p$ well-defined. Set $\mu'$ equal to $\mu$ but with fresh and distinct bound objects $\tilde{z}'$. As $\tilde{z}' \cap \mathrm{dom}(\Sigma) = \varnothing$, $\Gamma_+ \wr p \odot \sigma[\mathsf{obj}(\mu')]$ is also well-defined.

   (b) $\mu = \tau$. Let $\Gamma_+ = \Gamma$. Then $\Gamma_+ \wr \tau = \Gamma_+$ immediately implies $(\Gamma_+; P) \xrightarrow{\tau} (\Gamma_+; P')$.

3. Let $P \xrightarrow{\mu} P'$ be a transition whose subject port is $p$ with $p^{\omega} \succeq \Xi_{\mathrm{L}}$. Applying Lemma A.3.3, $P$ contains at most one prefix having $p$ in subject position, and if there is one it is replicated. By Lemma A.1.4, there is at least one prefix having $p$ in subject position. So we conclude $P \equiv (\boldsymbol{\nu}\tilde{z})\,(!\,(\boldsymbol{\nu}\tilde{z}')\,Q \mid R)$ with $R = a(\tilde{y}).R'$ (for $p = a$) or $R = \overline{a}\langle\tilde{x}\rangle.R'$ (for $p = \bar{a}$). The $\exists!Q$ s.t. $P \xrightarrow{\mu} Q$ condition is then immediately satisfied as $\mu$ must use that prefix, with the objects given by $\mu$.

□

The simple correctness lemma is now simply proven composing the above lemmas:

Let $\Gamma \vdash P$ and $(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'; P')$.

By the Subject Reduction Proposition, there is $\Gamma''$ such that $\Gamma'' \vdash P'$ and $\Gamma'' \succeq \Gamma'$. By the Local Correctness Lemma, $\mathsf{good}_{\#}(\Gamma''; P')$. By Lemma A.3.5, $\mathsf{good}_{\#}(\Gamma'; P')$ as well. Since this is valid for any transition sequence $\tilde{\mu}$, $\Gamma \models_{\#} P$.

## A.4 Proofs of Section 7

In this section we prove lemmas associated with structural analysis and the existential soundness proof.

### A.4.1 Subject Transitions (Lemma 7.4.5)

The transition put in communication a $\mathfrak{l}_I$-labelled guard with a $\mathfrak{l}_O$-labelled one in case neither is $\bullet$, or consumed a $\mathfrak{l}_I$-labelled (resp., $\mathfrak{l}_O$-) guard through a labelled transition, in which case we set $\mathfrak{l}_O$ (resp., $\mathfrak{l}_I$) to $\bullet$.

First assume $\mathfrak{p}$ is the free port $p$. Then $\mathfrak{p}' = \mathfrak{p}$.

Let $\rho$ be a runnable and complete strategy with $\rho \wr \pi \neq \bot$ such that $\mathsf{sub}_P(\rho) = p$, and set $\rho' = \rho \wr \pi$. We need to show that $\mathsf{sub}_{P'}(\rho') = p$ as well.

As $\rho' \neq \bot$, $\rho$ and $\pi$ don't contradict.

1. $\rho = \mathfrak{l}$.

   Then $\rho \wr \pi = \rho$.

   By non-contradiction, either $\mathfrak{l} \notin \{\mathfrak{l}_I, \mathfrak{l}_O\}$ or the $\mathfrak{l}$-tagged guard is replicated, so the $\mathfrak{l}$-guard is still available in $P'$ and $\mathsf{sub}_{P'}(\rho) = \mathsf{sub}_{P'}(\rho \wr \pi) = p$ as required.

2. $\rho = \tilde{\pi}.\,\mathfrak{l}$.

   Let $\pi_0 = (\mathfrak{l}_0|\rho_0)$ be the first step of $\tilde{\pi}$.

   This case is proven differently depending if $\pi_0$ matches $\pi$.

3. $\rho = \tilde{\pi}.\,\mathfrak{l}$ and $\pi_0$ does not match $\pi$.

   As $\pi$ and $\pi_0$ do not match, by non-contradiction, the $\mathfrak{l}_0$-guard must still be available unchanged in $P'$ (up to $\alpha$-renaming).

   Let $q$ be $\mathsf{sub}_P(\mathfrak{l})$ and $q'$ be $\mathsf{sub}_{P'}(\mathfrak{l})$. $\mathsf{sub}_P(\rho)$ and $\mathsf{sub}_{P'}(\rho')$ are respectively obtained by applying $\mathsf{subst}_P(\pi_i)$ and $\mathsf{subst}_{P'}(\pi'_i)$ in sequence from right to left. As the substitution only acts on free names and $p$ is free, we either have $q = p$, or one of the $\pi_i$ did the substitution $q \mapsto p$, because $\mathsf{obj}_P(\mathfrak{l}_i)[k] = \mathrm{n}(q)$ and $\mathsf{obj}_P(\rho_i)[k] = \mathrm{n}(p)$, for some index $k$.

   The $q = p$ case happens if and only if $q$ is not bound by any of its prefixes, which is preserved by the transition as the process in unchanged up to $\alpha$-renaming, so $\mathfrak{p} = \mathfrak{p}' = p$, as required.

   Assume instead $\mathsf{obj}_P(\mathfrak{l}_i)[k] = \mathrm{n}(q)$ and $\mathsf{obj}_P(\rho_i)[k] = \mathrm{n}(p)$, where $\mathfrak{l}_i$ binds $q$. Then $\alpha$-renaming preserves the index $k$ and the induction hypothesis preserves $\mathsf{obj}_P(\rho_i)[k] = \mathrm{n}(p)$, as $\mathrm{n}(p)$ is free, so $\mathfrak{p} = \mathfrak{p}' = p$, as required.

4. $\rho = \tilde{\pi}.\,\mathfrak{l}$ and $\pi_0$ matches $\pi$.

   Let $\bar{\mathfrak{l}}_0$ be such that $\{\mathfrak{l}_0, \bar{\mathfrak{l}}_0\} = \{\mathfrak{l}_I, \mathfrak{l}_O\}$. Then $\rho$ is transformed into $\rho'$ as follows: The $\pi_0$ prefix is dropped, every $\mathfrak{l}_i$ (including $\mathfrak{l}$) is replaced by $\mathfrak{l}'_i = \mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}_i)$ and every $\rho_i$ $(i \neq 0)$ is replaced by $\rho'_i = \rho_i \wr \pi$. The transition replaces a sub-process $G^{\mathfrak{l}_0}.Q$ by $\mathsf{mark}_{\bar{\mathfrak{l}}_0}(Q)\{^{\tilde{x}}/_{\mathsf{obj}(\mathfrak{l}_0)}\}$, where $\tilde{x}$ is one of $\mathsf{obj}(\mathfrak{l}_0)$, $\mathsf{obj}(\bar{\mathfrak{l}}_0)$ and $\mathsf{obj}(\mu)$, depending on whether $G$ is an input or an input, and whether $\bar{\mathfrak{l}}_0 = \bullet$ (there may be additional changes in the process, such as a similar reduction on a sub-process $G'^{\bar{\mathfrak{l}}_0}.Q'$, removal or expansion of bound names, and keeping a copy of those sub-processes if they are replicated).

   In particular each $\mathfrak{l}_i$ $(i > 0)$ both in $\rho$ and in $Q$ get replaced by $\mathfrak{l}'_i$.

   Three cases:

   - $\mathsf{sub}_P(\mathfrak{l}) = p$, i.e. $\mathfrak{l}$'s subject is free. See 5.
   - $\mathsf{sub}_P(\pi_1. \cdots . \mathfrak{l}) = p$, i.e. $\mathfrak{l}$'s subject is bound by an input contained inside $G$, and substituted to a free port by that input's communication partner. See 6.
   - $\mathsf{sub}_P(\pi_1. \cdots . \mathfrak{l}) = q$, i.e. $\mathfrak{l}$'s subject is bound but is substituted with a free port by $G$'s communication partner. See 7.

5. $\rho = \tilde{\pi}.\,\mathfrak{l}$, $\pi_0$ matches $\pi$, and $\mathsf{sub}_P(\mathfrak{l}) = p$.

   As in the $q = p$ case of point 3, $p$ is not bound by any of its prefixes. As the labelled transition system only substitutes bound names we have $\mathsf{sub}_{P'}(\mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l})) = p$ as well, which is still not bound by any of its prefixes so we get $\mathsf{sub}_{P'}(\rho') = p$ as required.

6. $\rho = \tilde{\pi}.\,\mathfrak{l}$, $\pi_0$ matches $\pi$, and $\mathsf{sub}_P(\pi_1. \cdots . \mathfrak{l}) = p$.

   Let $\mathsf{sub}_P(\mathfrak{l}) = q$. In order to compute $\mathsf{sub}_P(\pi_1. \cdots . \mathfrak{l})$, one applies all $\mathsf{subst}_P(\pi_1. \cdots . \pi_i)$ one by one with decreasing $i$ until one (say, $\pi_1. \cdots . \pi_j$, corresponding to some input guard $G_j$) substitutes $q$ with $p$. By hypothesis $j \neq 0$, $\mathrm{n}(q) = \mathsf{obj}_P(\mathfrak{l}_j)\,[\,k\,]$ for some $k$, $\mathsf{sub}(\mathfrak{l}_j)$ is an input and $\mathsf{obj}_P(\rho_j)\,[\,k\,] = \mathrm{n}(p)$.

   Let $\mathsf{sub}_{P'}(\mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l})) = q'$. It might be different from $q$ due to $\alpha$-renaming but we have $\mathrm{n}(q') = \mathsf{obj}_{P'}(\mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}_j))\,[\,k\,]$ because $\mathfrak{l}$ is contained in $G_j$'s continuation. As $p$ is free, induction hypothesis applies and $\mathsf{obj}_P(\rho_j) = \mathsf{obj}_{P'}(\rho_j \wr \pi)$, so the substitution works like before and $\mathsf{sub}_{P'}(\pi'_1. \cdots . \hat{\mathfrak{l}}) = p$, as required.

7. $\rho = \tilde{\pi}.\,\mathfrak{l}$, $\pi_0$ matches $\pi$, and $\mathsf{sub}_P(\pi_1. \cdots . \hat{\mathfrak{l}}) = q \in \mathrm{bn}(G)$.

   In this case $q$ got substituted to $p$ by $\pi_0$. This requires (Definition 7.2.1) $\pi_0$ to be doubly anchored, which in turn requires (for $\pi_0$ to match $\pi$) $\pi_0 = (\mathfrak{l}_0|\bar{\mathfrak{l}}_0)$. $\mathsf{subst}_P(\pi_0)$ is $\mathsf{obj}(\mathfrak{l}_0) \mapsto \mathsf{obj}(\bar{\mathfrak{l}}_0)$.

   In the process, $G^{\mathfrak{l}_0}.Q|G'^{\bar{\mathfrak{l}}_0}.Q'$ becomes $\mathsf{mark}_{\bar{\mathfrak{l}}_0}(Q\{^{\mathsf{obj}(\bar{\mathfrak{l}}_0)}/_{\mathsf{obj}(\mathfrak{l}_0)}\})|\mathsf{mark}_{\mathfrak{l}_0}(Q')$.

   Strategy subjects commute with substitution when free: If $\mathsf{sub}_P(\rho) = p$ then $\mathsf{sub}_{P\{^x/_y\}}(\rho) = p\{^x/_y\}$. In this case $\mathsf{sub}_{Q|\ldots}(\pi_1. \cdots . \mathfrak{l})\{^{\mathsf{obj}(\bar{\mathfrak{l}}_0)}/_{\mathsf{obj}(\mathfrak{l}_0)}\} = p$ implies $\mathsf{sub}_{\mathsf{mark}_{\mathfrak{l}_0}(Q\{^{\mathsf{obj}(\bar{\mathfrak{l}}_0)}/_{\mathsf{obj}(\mathfrak{l}_0)}\})\,|\,\ldots}(\pi'_1. \cdots . \mathsf{mark}_{\mathfrak{l}_0}(\mathfrak{l})) = p$.

   In other words $\mathsf{sub}_{P'}(\rho') = p$, as required.

Now let $\mathfrak{p} = \boldsymbol{\nu} p$, and let $P = (\boldsymbol{\nu}\tilde{z})\, P_0$.

If $\mu$ is a $\tau$ or an input then $\mathfrak{p}' = \mathfrak{p}$. If $\mu$ is an output, let $P_0 \xrightarrow{\mu_0} P_0'$ be the intermediate transition prior to the application of (Open) or (New) of the LTS. Claim: If $\mathrm{n}(p) = \mathsf{obj}(\mu_0)\,[\,k\,]$ then $\mathfrak{p}' = \mathsf{obj}(\mu)\,[\,k\,]$. Otherwise $\mathfrak{p}' = \mathfrak{p}$.

By Definition 7.2.1, $\mathsf{sub}_{(\boldsymbol{\nu}\tilde{z})\,P_0}(\rho) = (\boldsymbol{\nu} p)$ requires $\mathsf{sub}_{P_0}(\rho) = p$, otherwise the binding would be prefixed. Applying the reasoning done above for free $\mathfrak{p}$ we get $\mathsf{sub}_{P_0}(\rho) = \mathsf{sub}_{P_0'}(\rho') = p$.

In case the bound output $\mu$ did some $\alpha$-renaming on $\tilde{z}$ (say, $\{\tilde{y}/\tilde{z}\}$), we get $P' = (\boldsymbol{\nu}\tilde{y}')\,(P_0\{\tilde{y}/\tilde{z}\})$, and $\mathsf{sub}_{P'}(\rho') = (\boldsymbol{\nu}\tilde{y}')\,(p\{\tilde{y}/\tilde{z}\})$, for some $\tilde{y}' \subseteq \tilde{y}$. We have $\mathrm{n}(p)\{\tilde{y}/\tilde{z}\} \in \tilde{y}'$ precisely when the condition on $\mu$'s objects given above holds.

Now let $\mathfrak{p} = \hat{\pi}.\boldsymbol{\nu} p$, where $\hat{\pi} = (\hat{\mathfrak{l}}|\hat{\rho})$ matches $\pi$. Claim: if $\mathrm{n}(p) \in \mathsf{obj}_P(\hat{\mathfrak{l}})$, $\mathfrak{p}' = p\{{}^{\mathsf{obj}(\mu)}/_{\mathsf{obj}_P(\hat{\mathfrak{l}})}\}$ satisfies the requirements. Otherwise ($\mathrm{n}(p) \notin \mathsf{obj}_P(\hat{\mathfrak{l}})$) we have $\mathfrak{p}' = \boldsymbol{\nu} p$, modulo $\alpha$-renaming (done by the transition on $(\boldsymbol{\nu}\mathrm{n}(p))$ found at top-level in the process).

The $\mathfrak{p}' = \boldsymbol{\nu} p$ case is proved as part of the more general $\tilde{\pi}'.\boldsymbol{\nu} p$ later on. Assume $\mathrm{n}(p) \in \mathsf{obj}_P(\hat{\mathfrak{l}})$ and let $\mathsf{sub}_P(\rho) = \mathfrak{p}$.

1. $\rho = \mathfrak{l}$, by Definition 7.2.1, can't have a prefixed subject such as $\mathfrak{p}$.

2. $\rho = \tilde{\pi}'.\,\mathfrak{l}$.

   Let $q = \mathsf{sub}_P(\mathfrak{l})$. As $\mathfrak{p} \neq q$, $q$ must be bound by one of its prefixes, say $\mathfrak{l}_j$. Two cases: 3. $\mathrm{n}(q) \in \mathrm{bn}(\mathfrak{l}_j)$ and $\mathfrak{l}_j$ is either an output or a singly-anchored input, or $\mathfrak{l}_j$'s continuation binds $q$. 4. $\mathfrak{l}_j$ is a doubly-anchored input and $\mathrm{n}(q) \in \mathsf{obj}_P(\mathfrak{l}_j)$.

3. $\rho = \tilde{\pi}'.\,\mathfrak{l}$. $\mathrm{n}(q) \in \mathrm{bn}(\mathfrak{l}_j)$ and $\mathfrak{l}_j$ is either an output or a singly-anchored input, or $\mathfrak{l}_j$'s continuation binds $q$.

   Following Definition 7.2.1, $\mathsf{sub}_P(\rho_j.\,\cdots.\,\mathfrak{l}) = \pi_j.\boldsymbol{\nu} q$, and then all subsequent $\pi_i$ $(i < j)$ get added to that bound port, so we get $\mathsf{sub}_P(\rho) = \pi_0.\,\cdots.\pi_j.\boldsymbol{\nu} q$. By hypothesis $\mathsf{sub}_P(\rho) = \hat{\pi}.\boldsymbol{\nu} p$ so we conclude $j = 0$, $\pi_0 = \hat{\pi}$ and $p = q$.

   As $\pi$ matches $\hat{\pi}$, $\rho \wr \pi = \pi_1' \ldots \mathfrak{l}_j'$ where $\pi_i' = (\mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}_i)|\rho_i \wr \pi)$, $\bar{\mathfrak{l}}_0$ being $\mathfrak{l}_0$'s communication partner according to $\pi$.

   The process $P$, as $\rho$ is runnable, contains $G^{\mathfrak{l}_0}.Q$, where $Q$ contains $\mathfrak{l}$, and $\mathfrak{l}$'s subject $q$ is free in $Q$. After the transition ($\pi$ puts $\mathfrak{l}_0$ in communication with $\bar{\mathfrak{l}}_0$) that part of the process becomes $\mathsf{mark}_{\bar{\mathfrak{l}}_0}(Q)\{{}^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}$ in $P'$. We made the assumption $\mathrm{n}(q) = \mathrm{n}(p) \in \mathsf{obj}_P(\hat{\mathfrak{l}}) = \mathsf{obj}_P(\mathfrak{l}_0)$, so $\mathsf{sub}_{P'}(\mathfrak{l}) = q\{{}^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\} = p\{{}^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}$, as required (remember that $p = q$).

4. $\rho = \tilde{\pi}'.\,\mathfrak{l}$. $\mathfrak{l}_j$ is a doubly-anchored input and $\mathrm{n}(q) \in \mathsf{obj}_P(\mathfrak{l}_j)$.

   The proof of point 6 on page 150 (for $\mathfrak{p}$ free) applies here as well: $\mathsf{sub}_P(\mathfrak{l})$ is replaced by $\hat{\pi}.\boldsymbol{\nu} p$ which, by induction hypothesis, becomes $q\{{}^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}$ after the transition.

Now let $\mathfrak{p}$ be the bound sequence $\tilde{\pi}^*.\boldsymbol{\nu} p$, such that $\tilde{\pi}^*$ either has more than one step, or has a single step $\pi_0^*$ that either does not match $\pi$, or is such that $\mathrm{n}(p) \notin \mathsf{obj}_P(\mathfrak{l}_0^*)$. Then $\mathfrak{p}' = \mathfrak{p} \wr \pi$, where $\wr$ is defined as in Definition 7.4.2 and $\mathsf{mark}$ leaves bound names $\boldsymbol{\nu} p$ unchanged (up to $\alpha$-renaming — the last $\pi_j^*$ uniquely identifies in the process a binder of $\mathrm{n}(p)$, and if the transition $\alpha$-renames $\mathrm{n}(p)$, the corresponding change should be applied in $\mathfrak{p}'$).

1. $\rho = \mathfrak{l}$ is contradictory as before as its subject can't be $\mathfrak{p}$.

2. $\rho = \tilde{\pi}.\,\mathfrak{l}$

   Similarly to the $\mathfrak{p} = \hat{\pi}.\,\mathfrak{l}$ case, we distinguish whether $\mathsf{sub}(\mathfrak{l})$ gets bound (in which case $\rho = \tilde{\pi}^*.\,\pi_{j+1}.\,\cdots.\,\mathfrak{l}$ with $\pi_j = \pi_j^*$ binding $\mathsf{sub}_P(\mathfrak{l})$ where $\pi_j$ can only be a doubly-anchored input if its strategy is $\rho_j = \bullet$), or substituted (in which case the induction hypothesis applies as usual).

   We assume the former, as the latter has been covered already.

3. $\rho = \tilde{\pi}.\,\mathfrak{l}$. $\forall i \leq j : \pi_i = \pi_i^*$. $\pi_j$ binds $\mathsf{sub}_P(\mathfrak{l})$. $\rho_j = \bullet$ *or* $\pi_j$ is not a doubly-anchored input. $\pi_0$ doesn't match $\pi$.

   As $\mathsf{sub}(\rho)$ binds $p$ rather than substituting it, $\mathsf{sub}_P(\mathfrak{l}) = p$. As $\pi$ doesn't match $\pi_0$ but doesn't contradict $\rho$, the $\mathfrak{l}_0$-guard $G$ and its continuation $Q$ are left unchanged by the transition, up to $\alpha$-renaming, in particular the events $\mathfrak{l}_i$ are left as is. Let $\mathsf{sub}_{P'}(\mathfrak{l}) = p'$.

   As $\pi_0$ and $\pi$ do not match, $\rho \wr \pi = \rho' = \tilde{\pi}'.\,\mathfrak{l}$ where $\pi_i' = (\mathfrak{l}_i | \rho_i \wr \pi)$.

   As $G^{\mathfrak{l}_0}.Q$ is preserved in $P'$, $p'$ is not bound by any prefix $\pi_i'$ with $i > j$. It is bound (not substituted) by $\pi_j'$ because $\rho_j' = \bullet \iff \rho_j = \bullet$ and anchoring is preserved.

   The subject $\mathsf{sub}_{P'}(\rho')$ is therefore $\mathfrak{p}' = \pi_0'.\,\cdots.\,\pi_j'.\,\boldsymbol{\nu} p'$, as required.

4. $\rho = \tilde{\pi}.\,\mathfrak{l}$. $\forall i \leq j : \pi_i = \pi_i^*$. $\pi_j$ binds $\mathsf{sub}_P(\mathfrak{l})$. $\rho_j = \bullet$ *or* $\pi_j$ is not a doubly-anchored input. $\pi_0$ matches $\pi$. By $\rho$-runnability $P$ contains a process $G^{\mathfrak{l}_0}.Q$ that becomes $Q' = \mathsf{mark}_{\bar{\mathfrak{l}}_0}(Q)\{^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}$ (as in point 3 of case $\mathfrak{p} = \boldsymbol{\nu} p$ on page 151).

   As in the previous case, $p = \mathsf{sub}_P(\mathfrak{l})$ and let $p' = \mathsf{sub}_{P'}(\mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}))$.

   By hypothesis on $\mathfrak{p}$, at least one of these three conditions hold:

   - $j > 0$. See 5.
   - $\pi_0^*$ doesn't match $\pi$. Directly contradicts "$\pi_0$ matches $\pi$".
   - $\mathrm{n}(p) \notin \mathrm{bn}(\mathfrak{l}_0^*)$. See 6.

5. $\rho = \tilde{\pi}.\,\mathfrak{l}$. $\forall i \leq j : \pi_i = \pi_i^*$. $\pi_j$ binds $\mathsf{sub}_P(\mathfrak{l})$. $\rho_j = \bullet$ *or* $\pi_j$ is not a doubly-anchored input. $\pi_0$ matches $\pi$. $j > 0$.

   By the guarding constraints given by runnability, $\mathfrak{l}_j$ is contained in $Q$ and becomes $\mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}_j)$. As $p$ is bound by $\mathfrak{l}_j$ in $Q$, $p'$ must be bound by $\mathfrak{l}_j' = \mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}_j)$ in $Q'$, and so $\mathsf{sub}_{P'}(\pi_j'.\,\pi_{j+1}'.\,\cdots.\,\mathfrak{l}') = \boldsymbol{\nu} p'$, so we get $\mathsf{sub}_{P'}(\rho') = \mathsf{sub}_{P'}(\pi_1'.\,\cdots.\,\pi_j'.\,\cdots.\,\mathfrak{l}') = \pi_1'.\,\cdots.\,\pi_j'.\,\boldsymbol{\nu} p'$, as required.

6. $\rho = \tilde{\pi}.\,\mathfrak{l}$. $\forall i \leq j : \pi_i = \pi_i^*$. $\pi_j$ binds $\mathsf{sub}_P(\mathfrak{l})$. $\rho_j = \bullet$ *or* $\pi_j$ is not a doubly-anchored input. $\pi_0$ matches $\pi$. $\mathrm{n}(p) \notin \mathrm{bn}(\mathfrak{l}_0^*)$.

   As the $j > 0$ case got covered in the previous case, let $j = 0$, i.e. $\mathfrak{p} = \pi_0.\,\boldsymbol{\nu} p$ and $\mathsf{sub}_P(\pi_1.\,\cdots.\,\mathfrak{l}) = p$. As $\mathrm{n}(p) \notin \mathrm{bn}(\mathfrak{l}_0)$, we must have $Q = (\boldsymbol{\nu} \tilde{z})\, Q_0$ with $\mathrm{n}(p) \in \tilde{z}$ and $p$ free in $Q_0$.

   After the transition, $G^{\mathfrak{l}_0}.(\boldsymbol{\nu} \tilde{z})\, Q_0$ becomes $\mathsf{mark}_{\bar{\mathfrak{l}}_0}((\boldsymbol{\nu} \tilde{z}')\, Q_0\{^{\tilde{z}'}/_{\tilde{z}}\})\{^{\tilde{\mu}}/_{\mathsf{obj}(G)}\}$, with $p' = p\{^{\tilde{z}'}/_{\tilde{z}}\}$ (in other words the transition $\alpha$-renames $\tilde{z}$ to $\tilde{z}'$). As $p$ is free in $Q_0$, $p'$ is free in $Q_0\{^{\tilde{z}'}/_{\tilde{z}}\}$, so we get $\mathsf{sub}_{P'}(\rho') = \boldsymbol{\nu} p'$. As $\pi_0$ matches $\pi$, $\pi_0.\,\boldsymbol{\nu} p' \wr \pi = \boldsymbol{\nu} p'$ so we are done.

### A.4.2 Completeness of Strategies (Lemma 7.4.6)

The construction of $\rho$ from $\rho'$ is the same in all cases so we give it first.

Let $\pi = (\mathfrak{l}_I|\mathfrak{l}_O)$. The transition transforms a process sub-term $G_I{}^{\mathfrak{l}_I}.Q_I$ into $Q'_I = \mathsf{mark}_{\mathfrak{l}_O}(Q_I)\{^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G_I)}\}$ and/or $G_O{}^{\mathfrak{l}_O}.Q_O$ into $Q'_O = \mathsf{mark}_{\mathfrak{l}_I}(Q_O)$ (modulo $\alpha$-renaming). The "and/or" is resolved by checking if $\mu$ is an input (only produce $Q'_I$), an output (only $Q'_O$) or a $\tau$ (both $Q'_I$ and $Q'_O$).

Let $\rho' = \pi'_1.\,\pi'_2.\,\cdots.\,\mathfrak{l}_n$ with $\pi_i \in \{(\mathfrak{l}'_i|\rho'_i), (\mathfrak{l}'_i|\rho'_i]\}$.

If $\mathfrak{l}'_1$ occurs in $Q'_O$ (respectively, $Q'_I$), then for all $i$, $\mathfrak{l}'_i = \mathsf{mark}_{\mathfrak{l}_O}(\mathfrak{l}_i)$ (respectively, $\mathfrak{l}'_i = \mathsf{mark}_{\mathfrak{l}_I}(\mathfrak{l}_i)$), for some $\mathfrak{l}_i$. if $\mathfrak{l}'_1$ occurs in neither, set $\mathfrak{l}_i = \mathfrak{l}'_i$ for all $i$.

If $\mathfrak{l}'_1$ occurs in $Q'_I$, set $\pi_0 = \pi$. If $\mathfrak{l}'_1$ occurs in $Q'_O$, set $\pi_0 = \bar{\pi}$. Otherwise (to avoid a multiplication of otherwise similar cases) we'll say $\pi_0$ is "neutral" in the sense that $\pi_0.\,\rho \overset{\text{def}}{=} \rho$.

Apply this $\rho' \mapsto \rho$ transformation inductively (for the same transition $\mu$ and step $\pi$) to obtain $\rho_i$, for all $i$ s.t. $\rho'_i \neq \bullet$. The remaining $\rho_i$ are filled in for increasing values of $i$:

Let $\rho'_i = \bullet$. If $\mathsf{sub}_P(\pi_0.\,\pi_1.\,\cdots.\,\mathfrak{l}_i)$ is a free port, set $\rho_i = \bullet$ as well. Otherwise (the steps from $\pi_1$ to $\pi_{i-1}$ can't bind $\mathsf{sub}_P(\mathfrak{l}_i)$ as $Q'_I$ and $Q'_O$ were obtained from $Q_I$ and $Q_O$ by renaming that avoids capture), $G_I$ (or $G_O$) binds that port. Let $q$ be such that $\mathsf{obj}(G_I)\,[\,q\,] = \mathsf{sub}_P(\mathfrak{l}_i)$ (respectively, $\mathsf{obj}(G_O)\,[\,q\,] = \mathsf{sub}_P(\mathfrak{l}_i)$). Set $\rho_i = \bar{\pi}\,[\,q\,]$ (respectively, $\pi\,[\,q\,]$). Note that in both cases $\rho_i$ is of the form $(\bullet|\mathfrak{l})\,[\,q\,]$ with $\mathfrak{l} \in \{\mathfrak{l}_I, \mathfrak{l}_O\}$.

The strategy $\rho$ is then equal to $\pi_0.\,\pi_1.\,\cdots.\,\mathfrak{l}_n$ where $\pi_i = (\mathfrak{l}_i|\rho_i)$ for $i > 0$.

In case $\rho'$ was of the form $(\bullet|\rho'_0)\,[\,p\,]$, transform $\rho'_0$ into $\rho_0$ following the above procedure and set $\rho = (\bullet|\rho_0)\,[\,p\,]$.

The reader may want to verify that the above construction implies $\rho \wr \pi = \rho'$ in all cases.

To verify guarding constraints on $\rho$ for $P$, assume $\mathfrak{l}'_1$ is neither in $Q'_I$ nor in $Q'_O$. Then the sequence $\mathfrak{l}'_1, \ldots, \mathfrak{l}'_n$ has each event guard the next in $P'$, with $\mathfrak{l}'_1$ at top-level, and therefore the sequence $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$ also has each event guard the next in $P$ with $\mathfrak{l}_1$ at top-level (remember that in this case $\forall i : \mathfrak{l}_i = \mathfrak{l}'_i$. If $\mathfrak{l}'_1$ is in $Q'_I$, the $\mathfrak{l}'_i$ sequence similarly satisfies guarding requirements with $\mathfrak{l}'_1$ at top-level in $P'$ and therefore in $Q'_I$. By the definition of $Q'_I$, $\mathfrak{l}_1$ is at top-level in $Q_I$ and all $\mathfrak{l}_{i+1}$ with $i \geq 1$ are guarded by $\mathfrak{l}_i$. As the first step of $\rho$ is $\pi_0 = \pi = (\mathfrak{l}_I|\mathfrak{l}_O)$ and $Q_I$ is the continuation of $G_I{}^{\mathfrak{l}_I}$, $\mathfrak{l}_0 = \mathfrak{l}_I$ is at top-level and guards $\mathfrak{l}_1$, as required. The $Q'_O$ case is similar, swapping $\mathfrak{l}_I$ and $\mathfrak{l}_O$.

We now show a $\mathfrak{p}' \mapsto \mathfrak{p}$ transformation that is consistent with $\rho' \mapsto \rho$ and satisfies the lemma requirements.

We treat all possible cases one by one, subdividing cases as needed. Each point starts with the hypotheses for the case followed by the proof for that case.

We first distinguish if $\mathfrak{p}'$ is free (case 1) or bound (case 6).

1. $\mathfrak{p}'$ is a free port $p'$.

   By hypothesis if $\mu$ is an input its objects must be fresh. If $\mu$ is an output, its bound objects must not be in $\mathrm{fn}(P)$, because of the side condition of the (PAR) LTS rule. Therefore, if $\mathrm{n}(p') \in \mathrm{bn}(\mu)$ then $\mathrm{n}(p')$ is not free in $P$ and $\mathfrak{p}$ must be bound (Case 2). Otherwise $\mathfrak{p} = p'$ as well, as shown in Case 5.

2. $\mathfrak{p}'$ is a free port $p'$. $\mathrm{n}(p') \in \mathrm{bn}(\mu)$.

   Let $\mathfrak{l}$ be such that $(\bullet|\mathfrak{l}) \in \{\pi, \bar{\pi}\}$ (we have $\pi = (\bullet|\mathfrak{l}_O)$ in case $\mu$ is an output and $\pi = (\mathfrak{l}_I|\bullet)$ in case $\mu$ is an input. $\mu = \tau$ is excluded as $\mathrm{bn}(\mu) \neq \varnothing$.)

   Let $q$ be such that $p' = \mathsf{obj}(\mu)\,[\,q\,]$ and set $p = \mathsf{obj}(G)\,[\,q\,]$ (where $G$ is the prefix in $P$ consumed by $\mu$, i.e. $G_I$ if $\mu$ is an input, $G_O$ otherwise).

   Then $\mathfrak{p} = (\mathfrak{l}|\bullet).\boldsymbol{\nu} p$ satisfies the requirements as we show now.

   Let $\rho' = \pi'_1.\,\cdots.\,\mathfrak{l}'_n$ be a strategy such that $\mathsf{sub}_{P'}(\rho') = p'$, and let $\rho$ be the strategy obtained as described earlier.

   As $\mathfrak{p}'$ is free, we either have $\mathsf{sub}_{P'}(\mathfrak{l}'_n) = p'$ (case 3) or $\mathsf{sub}_{P'}(\mathfrak{l}'_n) = p_0$ and one of the $\mathsf{subst}_{P'}(\pi'_i)$ substitutes $\mathrm{n}(p_0)$ to $\mathrm{n}(p')$ (case 4).

3. $\mathfrak{p}'$ is a free port $p'$. $\mathrm{n}(p') \in \mathrm{bn}(\mu)$. $\mathsf{sub}_{P'}(\mathfrak{l}'_n) = p'$.

   As $p'$ is fresh, $\mathfrak{l}_n$ must appear in the continuation $Q$ (one of $Q_I$ and $Q_O$) of $G$ and $\mathfrak{l}'_n$ in the corresponding process term $Q'$ in $P'$. So the $\rho' \mapsto \rho$ construction implies $\rho = (\mathfrak{l}|\bullet).\,\pi_1.\,\cdots.\,\mathfrak{l}_n$ and $\mathsf{sub}_p(\pi_1.\,\cdots.\,\mathfrak{l}_n) = p$. $\mathrm{n}(p)$ is bound in $G$ as it is bound in the transition label, so $\mathsf{sub}_P(\rho) = (\mathfrak{l}|\bullet).\boldsymbol{\nu} p$, as required.

4. $\mathfrak{p}'$ is a free port $p'$. $\mathrm{n}(p') \in \mathrm{bn}(\mu)$. $\mathsf{sub}_{P'}(\mathfrak{l}'_n) = p'_0$ and $\mathsf{subst}_{P'}(\pi'_j)$ substitutes $\mathrm{n}(p'_0)$ to $\mathrm{n}(p')$.

   So $\mathsf{obj}_{P'}(\mathfrak{l}'_j)\,[\,q\,] = p'_0$ and $\mathsf{obj}_{P'}(\rho'_j)\,[\,q\,] = p'$ for some $q$. By induction hypothesis there is $\rho_j$ satisfying the lemma conditions (where $\rho'$ and $\rho$ in the statement stand for $\rho'_j$ and $\rho_j$), so $\mathsf{obj}_P(\rho_j)q = \mathfrak{p}$.

   Let $\mathsf{sub}_{P'}(\mathfrak{l}'_n) = p_0$ (which may be distinct from $p'_0$ in case $\alpha$-renaming occurred). Then $\mathrm{n}(p_0)$ is not bound by any of the prefixes corresponding to $\pi_i$ with $j < i < n$ (as that property is preserved by $\alpha$-renaming and *capture-avoiding* substitution). For the same reasons $\mathrm{n}(p_0)$ is bound by $\mathfrak{l}_j$, so $\mathsf{sub}_P(\rho) = p_0\{^{\mathsf{obj}_P(\rho_j)}/_{\mathsf{obj}_P(\mathfrak{l}_j)}\} = \mathfrak{p}$, as required.

   Note that the proof of this case works every time a subject is captured by a $\mathsf{subst}(\pi_i)$-substitution so in the following cases we assume that no $\mathsf{subst}_{P'}(\pi'_i)$ captures $\mathsf{sub}_{P'}(\mathfrak{l}'_n)$.

5. $\mathfrak{p}'$ is a free port $p$. $\mathrm{n}(p) \notin \mathrm{bn}(\mu)$.

   In this case $\mathfrak{p} = \mathfrak{p}' = p$ satisfies the requirements (we write $p$ instead of $p'$ because there is no renaming involved but the reader may prefer to write $\mathfrak{p}' = p'$ and $\mathfrak{p} = p$, with of course $p = p'$).

   Let $\rho' = \pi'_1.\,\cdots.\,\mathfrak{l}'_n$ be such that $\mathsf{sub}_{P'}(\rho') = p$. So for all $0 < i < n$, $\mathfrak{l}'_i$ does not bind $\mathrm{n}(p)$. This is preserved by renaming so for all $0 < i < n$, $\mathfrak{l}_i$ does not bind $\mathrm{n}(p)$ and we get $\mathsf{sub}_P(\pi_1.\,\cdots.\,\mathfrak{l}_n) = p$. As $\mu$'s input or bound objects are fresh, they are necessarily distinct from $\mathrm{n}(p)$, so $\mathsf{sub}_P(\mathfrak{l}_n) = \mathsf{sub}_{P'}(\mathfrak{l}'_n)$ and either $\mathfrak{l}_1$ is at top-level (in which case we're done) or $\mathfrak{l}_1$ is guarded by one of $\mathfrak{l}_I$ and $\mathfrak{l}_O$ which, by hypothesis, doesn't bind $p$, so $\mathsf{sub}_P(\rho) = \mathsf{sub}_P(\pi_1.\,\cdots.\,\mathfrak{l}_n) = p$, as required.

6. $\mathfrak{p}'$ is bound.

   Examining the proof of Lemma 7.4.5, the only case in which $\mathfrak{p}' = \mathsf{sub}_{P'}(\rho')$ is bound requires $\mathfrak{p} = \mathsf{sub}_P(\rho)$ being bound as well, and satisfy $\mathfrak{p} \wr \pi = \mathfrak{p}'$.

Fix $\mathfrak{p}' = \pi'_1. \cdots . \pi'_j. \boldsymbol{\nu} p'$. Then $\mathfrak{p} = \pi_0. \cdots . \pi_j. \boldsymbol{\nu} p$ satisfies the requirements, where $p \mapsto p'$ corresponds to any $\alpha$-renaming occurring in the $P \xrightarrow{\mu;\pi} P'$ transition and $\pi_0$ is either "neutral" (when $\mu = \tau$, $\mathfrak{p}$ is really $\pi_1. \cdots . \pi_j. \boldsymbol{\nu} p$) or a step $(\mathfrak{l}|\bullet) \in \{\pi, \bar{\pi}\}$, just like described in the $\rho' \mapsto \rho$ mapping at the beginning of this proof. Note that $\mathfrak{p} \wr \pi = \mathfrak{p}'$.

Let $\rho'$ be a strategy with subject $\mathfrak{p}'$ and define $\pi'_i$, $\mathfrak{l}'_i$, $\rho'_i$ and their counterparts without a tick $'$ as in all previous cases. Note that the $\pi'_i$ for $i \leq j$ necessarily coincide with the ones occurring in $\mathfrak{p}'$, by the definition of $\mathsf{sub}_{P'}$. As in the previous cases $\pi'_j$ is the step with largest $j$ that binds $p'$, and we assume $\mathsf{subst}_{P'}(\pi'_j)$ doesn't capture it (if it does, refer to step 4). All this properties are preserved by renaming and marking, so $\pi_j$ is the step with largest $j$ that binds $p$, and $\mathsf{subst}_P(\pi_j)$ doesn't capture it. So we immediately get $\mathsf{sub}_P(\rho) = \pi_0. \pi_1. \cdots . \pi_j. \boldsymbol{\nu} p = \mathfrak{p}$, as required.

### A.4.3 Runnability Safety (Lemma 7.4.8)

First of all, $\Gamma'$ is elementary by definition of the $\searrow$ relation.

We first prove $\Gamma'$ is consistent before proceeding to completeness. We prove the lemma just for subjects, as the proof for targets is identical, just replacing $\mathsf{sub}_P$ by $\mathsf{trg}_{k,P}$ everywhere (and is valid, by virtue of target function commuting with substitution).

Let $\Gamma$'s local dependency network be $s_k \lhd \varepsilon : \rho$. Let $\pi = (\mathfrak{l}_I|\mathfrak{l}_O)$ be the step used to prove $(\Gamma; P) \xrightarrow{\mu}\searrow (\Gamma'; P')$ following Definition 7.4.3. Then that transition put in communication a $\mathfrak{l}_I$-labelled guard with a $\mathfrak{l}_O$-labelled one in case neither is $\bullet$, or consumed a $\mathfrak{l}_I$-labelled (resp., $\mathfrak{l}_O$-) guard through a labelled transition.

The strategy of $s_k$ in $\Gamma'$ is $\rho' = \rho \wr \pi$. We assume $\rho' \neq \bot$ otherwise $\Gamma' = \top$ which is vacuously consistent. This implies that $\pi$ doesn't contradict $\rho$. By hypothesis, $\rho$ is runnable. We show by induction on $\rho$'s structure that all conditions in Definition 7.2.3 are preserved in $\rho'$. By induction hypothesis, the sub-strategies of $\rho'$ are runnable.

There is a large number of cases that need to be proved separately.

1. $\rho = \mathfrak{s}$.

   By runnability it must be at top-level in $P$. If neither $\mathfrak{s} \cap \{\mathfrak{l}_I, \mathfrak{l}_O\} = \varnothing$ (seeing the sum $\mathfrak{s}$ as the set of its terms) then $\rho' = \mathfrak{s}$ and $\mathfrak{s}$ was not consumed by $\mu$, so it still is at top-level in $P'$. If $\mathfrak{s} \cap \{\mathfrak{l}_I, \mathfrak{l}_O\} = \mathfrak{l}$ then, by non-contradiction, $\mathfrak{s}$ must be replicated in $P$ and $\mathfrak{s} = \mathfrak{l}$, so it remains at top-level in $P'$ no matter what $\mu$ is doing.

2. $\rho = \tilde{\pi}. (\hat{\mathfrak{l}}|\hat{\rho}). \mathfrak{s}$ or $\rho = (\hat{\mathfrak{l}}|\hat{\rho}). \mathfrak{s}$.

   Let $\mathfrak{p} = \mathsf{sub}_P(\tilde{\pi}. \hat{\mathfrak{l}})$ and $\mathfrak{p}' = \mathsf{sub}_{P'}(\tilde{\pi}'. \hat{\mathfrak{l}}')$. Then:

   - $\hat{\mathfrak{l}}$ guards $\mathfrak{s}$.
   - If $\hat{\rho} = \bullet$: $\mathfrak{p} = p$ for some $p$ and $\Gamma\downarrow_p$. See 3.
   - If $\hat{\rho} \neq \bullet$: $\mathsf{sub}_P(\rho) = \bar{\mathfrak{p}}$. See 4.

   The first condition, that the $\hat{\mathfrak{l}}$ guards $\mathfrak{s}$ is proved differently depending if $\pi_0$ matches $\pi$ (point 6) or not (point 5).

3. $\rho = \tilde{\pi}.\,(\hat{\mathfrak{l}}|\bullet).\,\mathfrak{s}$ or $\rho = (\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$. $\mathfrak{p} = p$ for some $p$. $\Gamma \downarrow_p$.

   By Lemma 7.4.5, $\mathfrak{p}' = p$ as well, and, by non-contradiction with $\mathfrak{s}$, $\Gamma' \downarrow_p$ as well.

4. $\rho = \tilde{\pi}.\,(\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$ or $\rho = (\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$. $\hat{\rho} \neq \bullet$. $\mathsf{sub}_P(\rho) = \bar{\mathfrak{p}}$.

   By Lemma 7.4.5, $\mathsf{sub}_P(\hat{\rho}) = \bar{\mathfrak{p}}$ implies $\mathsf{sub}_{P'}(\hat{\rho} \wr \pi) = \overline{\mathfrak{p}'}$.

   Let $\pi_0 = (\mathfrak{l}_0|\rho_0)$ be the first step of $\tilde{\pi}$, (or be $(\hat{\mathfrak{l}}|\hat{\rho})$ in case $\tilde{\pi}$ is empty).

5. $\rho = \tilde{\pi}.\,(\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$ or $\rho = (\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$. $\pi_0$ does not match $\pi$.

   $\rho'$ is equal to $\rho$ in the $\mathfrak{l}_i$, and the $\rho_i$ are replaced by $\rho_i \wr \pi$. The sequence of $\mathfrak{l}_i$ is therefore preserved by transition, and, by non-contradiction, $\mathfrak{l}_0$ and the process it guards is preserved by $\mu$. In particular, the $\hat{\mathfrak{l}}$ guards $\mathfrak{s}$ condition is preserved.

6. $\rho = \tilde{\pi}.\,(\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$ or $\rho = (\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$. $\pi_0$ matches $\pi$.

   Let $\bar{\mathfrak{l}}_0$ be such that $\{\mathfrak{l}_0, \bar{\mathfrak{l}}_0\} = \{\mathfrak{l}_I, \mathfrak{l}_O\}$. Then $\rho$ is transformed into $\rho'$ as follows: The $\pi_0$ prefix is dropped, every $\mathfrak{l}_i$ (including $\mathfrak{s}$ and, if applicable, $\hat{\mathfrak{l}}$) is replaced by $\mathfrak{l}'_i = \mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}_i)$ and every $\rho_i$ $(i \neq 0)$ is replaced by $\rho'_i = \rho_i \wr \pi$. The transition replaces a sub-process $G^{\mathfrak{l}_0}.Q$ by $\mathsf{mark}_{\bar{\mathfrak{l}}_0}(Q)\{^{\tilde{x}}/_{\mathsf{obj}(\mathfrak{l}_0)}\}$, for some $\tilde{x}$. In particular every $\mathfrak{l}_i$ $(i > 0)$, including $\hat{\mathfrak{l}}$ and $\mathfrak{s}$, is replaced in both $Q$ and $\rho$ by $\mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}_i)$. The two following cases cover the two possible forms of $\rho$.

7. $\rho = \tilde{\pi}.\,(\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$. $\pi_0$ matches $\pi$.

   $\hat{\mathfrak{l}}$ guarding $\mathfrak{s}$ in $Q$ implies that $\hat{\mathfrak{l}}'$ guards $\mathfrak{l}'$ in $Q'$, as required.

8. $\rho = (\hat{\mathfrak{l}}|\hat{\rho}).\,\mathfrak{s}$. $\hat{\pi}$ matches $\pi$.

   $\rho' = \mathfrak{l}'$. As $\hat{\mathfrak{l}}$ guards $\mathfrak{s}$ in $P$, $\mathfrak{s}$ is at top-level in $Q$ and $\mathfrak{l}'$ is at top-level in $Q'$, so at top-level in $P'$ as well, as required.

We now show that completeness is preserved by the transition. Let $\Phi = \bigvee_{i \in I} \Phi_i$. As $\wr$ is a logical homomorphism, $\Phi' = \bigvee_{i \in I} \Phi'_i$ where $\Phi'_i = \Phi_i \wr \pi$. We set once more $\pi = (\mathfrak{l}_I|\mathfrak{l}_O)$.

A key part of proving that is the following corollary of Lemma 7.4.6: Let $\rho'$ be a selection strategy for $P'$. Then there is a selection strategy $\rho$ for $P$ such that $\rho \wr \pi = \rho'$.

The construction of $\rho$ from $\rho'$, $\mu$ and $\pi$ is given in the proof of Lemma 7.4.6. We show that $\rho$ is runnable if $\rho'$ is. The guarding constraints have already been shown in the lemma but we still have to show that $(\mathfrak{l}_i|\rho_i)$-steps satisfy the complementarity constraint when $\rho_i \neq \bullet$, and that $(\mathfrak{l}_i|\bullet)$-steps satisfy the free name requirements.

We work by induction on the weight of $\rho'$.

Let $\rho' = \pi'_1.\,\cdots.\,\mathfrak{l}'_n$ (for a selection strategy whose final step is a pair the proof is the same, just ignoring the $\mathfrak{l}'_n$ and requiring $n > 1$). Let $\rho = \pi_0.\,\pi_1 \cdots.\,\mathfrak{l}_n$ be obtained from $\rho'$ as given in the proof of Lemma 7.4.6.

We treat differently the "base case" $n = 1$ (i.e. $\rho' = \mathfrak{l}'_1$, Case 1) and the "step case" $n > 1$ (Case 2).

1. $\rho' = \mathfrak{l}'_1$.

   If $\pi_0$ is "neutral", $\rho = \mathfrak{l}_1$ and there's nothing to show (we already showed as part of Lemma 7.4.6 that $\mathfrak{l}_1$ is at top-level in $P$).

   Otherwise $\rho \in \{(\mathfrak{l}_I|\mathfrak{l}_O), (\mathfrak{l}_O|\mathfrak{l}_I)\}$. If neither is $\bullet$, $\mu = \tau$ and the transition was proved from the (A-Com)-rule of the LTS which requires the subjects of communicating guards to be complements, i.e. $\mathsf{sub}_P(\mathfrak{l}_I) = a$ and $\mathsf{sub}_P(\mathfrak{l}_O) = \bar{a}$ for some $a$ so we're done.

   If $\pi_0 = (\mathfrak{l}|\bullet)$ then $\mu \neq \tau$ has subject $p = \mathsf{sub}_P(\mathfrak{l})$ and $\Gamma'$ being well-defined requires $\Gamma \wr p$ being defined as well, from the definition of the $\wr$ operator, i.e. $p$ is observable, as required.

2. $\rho' = \pi'_1. \cdots .\mathfrak{l}'_n$, $n > 1$.

   Following the usual naming convention we have $\pi'_{n-1} = (\mathfrak{l}'_{n-1}|\rho'_{n-1})$. We treat $\rho'_{n-1} = \bullet$ (Case 3) and $\rho'_{n-1} \neq \bullet$ (Case 6) differently.

3. $\rho' = \pi'_1. \cdots .\mathfrak{l}'_n$, $n > 1$. $\rho'_{n-1} = \bullet$.

   As $\rho'$ is runnable, $\mathsf{sub}_{P'}(\pi'_1. \ldots \mathfrak{l}'_{n-1})$ is free in $P'$ (let's call it $p'$) and $\Gamma'$-observable.

   Applying Lemma 7.4.6, $\mathfrak{p} = \mathsf{sub}_P(\pi_0. \pi_1. \ldots \mathfrak{l}_{n-1})$ is either free and equal to $p'$ (Case 4), or is bound and equal to $(\mathfrak{l}|\bullet).\boldsymbol{\nu}p$ for some $\mathfrak{l}$ given by $\pi$, and $p$ (Case 5).

4. $\rho' = \pi'_1. \cdots .\mathfrak{l}'_n$, $n > 1$. $\rho'_{n-1} = \bullet$. $\mathfrak{p}$ is a free port $p$.

   As shown in Lemma 7.4.6 we have $p = p'$ and $p$ is necessarily observable in $\Gamma$ as $\mu$ is either $\tau$ (in which case $\Gamma = \Gamma'$) or an input that doesn't bind $\mathrm{n}(p)$.

5. $\rho' = \pi'_1. \cdots .\mathfrak{l}'_n$, $n > 1$. $\rho'_{n-1} = \bullet$. $\mathfrak{p} = (\mathfrak{l}|\bullet).\boldsymbol{\nu}p$.

   As shown in Lemma 7.4.6, $\mathfrak{p}$ bound can only become $\mathfrak{p}'$ free if $\mu \neq \tau$. So $p = \mathsf{obj}_P(\mathfrak{l})\,[\,q\,]$ for some $q$ and $p' = \mathfrak{p}' = \mathsf{obj}(\mu)\,[\,q\,]$.

   The $\rho' \mapsto \rho$-construction sets $\rho_{n-1} = (\bullet|\mathfrak{l})\,[\,\bar{q}\,]$ in this case. We then have $\mathsf{sub}_P(\rho_{n-1}) = (\mathfrak{l}|\bullet).\boldsymbol{\nu}\bar{p} = \bar{\mathfrak{p}}$, as required.

6. $\rho' = \pi'_1. \cdots .\mathfrak{l}'_n$, $n > 1$. $\rho'_{n-1} \neq \bullet$.

   By runnability, $\mathsf{sub}_{P'}(\rho'_{n-1}) = \overline{\mathfrak{p}'}$. Having $\mathfrak{p} = \mathsf{sub}_P(\pi_0. \cdots .\mathfrak{l}_{n-1}) = \mathfrak{p}$, noting that $\rho_n$ and $\pi_0. \cdots .\mathfrak{l}_{n-1}$ have been obtained from $\rho'_{n-1}$ and $\pi_1.' \cdots .\mathfrak{l}'_{n-1}$ following the same $\rho' \mapsto \rho$-construction as in Lemma 7.4.6 we have $\mathsf{sub}_P(\rho_{n-1}) = \bar{\mathfrak{p}}$, as required.

### A.4.4 Strategy Application (Lemma 7.4.9)

The conclusion can be obtained in three different ways:

1. $(\Gamma; P)$ is immediately correct.

2. $(\Gamma'; P')$ is immediately correct.

3. $(\Gamma'; P')$ is not immediately correct but has a weight strictly less than $(\Gamma; P)$.

Let $\Gamma$'s local dependency network be $s_k \lhd \varepsilon\colon \rho$. We proceed by induction on $\mathsf{wt}(\rho)$, and will have to consider all three cases above when using the induction hypothesis.

If $\rho = \mathfrak{s} = \sum_i \mathfrak{l}_i$ then $\mathfrak{s}$ is at top-level in $P$, i.e. $P \equiv (\boldsymbol{\nu}\tilde{a})\,(\sum_i {G_i}^{\mathfrak{l}_i}.Q_i \mid R)$, where $\sum_i \mathsf{sub}(G_i) = s$. By consistency of $\mathfrak{s}$, $\mathsf{trg}_{k,P}(\mathfrak{s}) = s_k$, and by definition of target functions (Definition 7.2.2), the $k$-elementary rules type $\sum_i {G_i}^{\mathfrak{l}_i}.Q_i$ as $s_k$. By definition of elementary rules (Definitions 4.4.1 and 4.4.3), and $k$ being existential (so the composition with $R$ preserves correctness), $\mathsf{good}_k(s \lhd \top, (\Gamma; P))$, so $\Gamma$ is immediately correct.

If $\rho = (\rho_0|\bullet)\,[\,s'\,]$, let $p_0 = \mathsf{sub}(\rho_0)$. Set $\Gamma_0$ to $\Gamma$ but with local component $p_{0k} \lhd \varepsilon : \rho_0$. As $\Gamma$ is consistent and complete, so is $\Gamma_0$, and the induction hypothesis applies. Case 1: $\Gamma_0$ is immediately correct so $p_0$ is at top-level and there is a transition $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ with $\mathsf{sub}(\mu) = p_0$ and, by Definition 7.2.1 and consistency of $\Gamma$, $\mathsf{obj}(\mu)\,[\,s'\,] = s$, so $\Gamma \wr \mu$ drops activeness on $s$ (See the definition of $\Gamma \wr \mu$ in Section 6.2), rendering $(\Gamma'; P')$ immediately correct. Cases 2 and 3: there is a transition $(\Gamma_0; P) \xrightarrow{\mu} (\Gamma'_0; P')$ satisfying the requirements in the Lemma statement. Then $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$. Let the local component of $\Gamma'_0$ be $p_{0k} \lhd \varepsilon' : \rho'_0$. Then the local component of $\Gamma'$ is $s_k \lhd \varepsilon' : (\rho'_0|\bullet)\,[\,s'\,]$. As (by induction hypothesis) $\mathsf{wt}(\rho'_0) < \mathsf{wt}(\rho_0)$, $\mathsf{wt}((\rho'_0|\bullet)\,[\,s'\,]) < \mathsf{wt}((\rho_0|\bullet)\,[\,s'\,])$, as required.

Now assume $\rho = (\mathfrak{l}_0|\rho_0).\,\rho_1$ with $\rho_0 \neq \bullet$. Let $\mathsf{sub}_P(\mathfrak{l}_0) = p_0$. Then $\rho_0$ is a runnable strategy for $\bar{p}_0$ and the induction hypothesis applies. Case 1: $\bar{p}_0$ is at top-level in $P'$ so there is a transition $(\Gamma; P) \xrightarrow{\mu'} (\Gamma'_0; P'_0)$ where $\mu'$ has $\bar{p}_0$ in subject position. Applying the (Com) rule of the LTS the $\mu'$ transition can be replaced by a $\tau$-transition additionally consuming $\mathfrak{l}_0$, let that transition be $(\Gamma; P) \xrightarrow{\tau} (\Gamma'; P')$. Then the local component of $\Gamma'$ is $s_k \lhd \varepsilon\colon \rho_1$. If $\bar{p}_0$ is an object of the $\mu'$ transition, $\bar{p}_{0k}$ will be provided by the environment and one can do (after $\mu'$) one labelled transition consuming $\mathfrak{l}_0$, like in the $\rho = \mathfrak{s}$ case above, and again we're back to the above case.

Case 2 and case 3: Let $\Gamma_0$ be $\Gamma$ but with local component $\bar{p}_{0k} \lhd \varepsilon\colon \rho_0$. By induction hypothesis there is a transition $(\Gamma_0; P) \xrightarrow{\mu} (\Gamma'_0; P')$ as in the Lemma statement. Let $(\Gamma; P) = \xrightarrow{\mu} (\Gamma'; P')$ be the corresponding transition (i.e. just like $\Gamma'_0 = \Gamma_0 \wr \mu$, $\Gamma' = \Gamma \wr \mu$). The local component of $\Gamma'_0$ being $p_{0k} \lhd \varepsilon\colon \rho'_0$, we have $s_k \lhd \varepsilon\colon (\mathfrak{l}_0|\rho'_0).\,\rho_1$ as local component of $\Gamma'$.

If $\rho = (\mathfrak{l}_0|\bullet).\,\rho_1$, let $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ be a transition consuming $\mathfrak{l}_0$. Then $p$'s strategy in $\Gamma'$ is $\rho_1$, which has a weight lower than $\rho$. By runnability of $\rho$ and the definition of the $\mathsf{dep}_P$ operator, $\bar{p}_{0k}$ must provided by the environment.

The $\rho = \tilde{\pi}_1 \lightning (\tilde{\pi}_2)\delta$ case is essentially identical to the above ones, by focusing on the $\tilde{\pi}_1$ part and leaving the rest unchanged.

### A.4.5 Completeness and Correctness (Lemma 7.4.10)

As $\tilde{\mu}_0$ is empty, $\Gamma_0 \searrow \Gamma'_0$, so $(\Gamma_0; P_0)$ being consistent and complete implies $(\Gamma'_0; P_0)$ is consistent and complete, and $\Gamma'_0$ has a weight lower or equal to that of $\Gamma_0$.

By repeated application of Lemma 7.4.8 on the sequence $(\Gamma_i; P_i) \xrightarrow{\tilde{\mu}_i}\searrow (\Gamma'_i; P'_i)$, if $(\Gamma_i; P_i)$ is consistent and complete then $(\Gamma'_i; P'_i)$ is consistent and complete as well, and $\Gamma'_i$ has a weight smaller than or equal to $\mathsf{wt}(\Gamma_i)$.

The strategy $f$ is defined following Lemma 7.4.9, producing, for consistent and complete but not immediately correct typed processes $(\Gamma'_i; P'_i)$, transitions $(\Gamma'_i; P'_i) \xrightarrow{\mu} (\Gamma_{i+1}; P_{i+1})$, such that $\mathsf{wt}(\Gamma_i) > \mathsf{wt}(\Gamma_{i+1})$.

For all $i < j$: $\mathsf{wt}(\Gamma'_i) > \mathsf{wt}(\Gamma'_j)$. As weight can't be negative, there is a value of $n$ as in Definition 5.2.6 on page 54 of at most $\mathsf{wt}(\Gamma'_0)$ such that $i > n$ implies $(\Gamma_i; P_i)$ is immediately correct.

## A.4.6 Reduction and Composition Preserve consistency (Lemmas 7.5.6, 7.5.7)

We show that all four transformations (7.5), (7.6), (7.7) and (7.8) given in Definitions 7.5.1 and 7.5.3 preserve runnability.

Note that a strategy of the form $\mathfrak{s}$ can't be altered or produced by the rules because it doesn't match any of them, on the left or right of the $\mapsto$ symbol. So we only consider sub-strategies of the form $\tilde{\pi}.\,\mathfrak{s}$ or $\pi\,[\,s'\,]$, and show that the label-guarding property is preserved and subjects of newly introduced $(\mathfrak{l}|\rho)$-pairs are complements, as required. Additionally we show that $(s_k \cdots : \rho) \mapsto (s_k \cdots : \rho')$ implies $\mathsf{sub}(\rho) = \mathsf{sub}(\rho')$, i.e. $\mathsf{sub}(\rho') = s$ as is required for consistency.

(7.5) Calling the "main event sequence" of a strategy $(\mathfrak{l}_0|\rho_0).\,(\mathfrak{l}_1|\rho_1).\,\cdots.\,\mathfrak{s}$ the sequence $(\mathfrak{l}_0, \mathfrak{l}_1, \ldots, \mathfrak{s})$, runnability requires $\mathfrak{l}_0$ to be at top-level and every $\mathfrak{l}_i$ with $i < n$ to guard $\mathfrak{l}_{i+1}$ ($\mathfrak{s}$ in case $i = n-1$). That sequence is preserved by rule (7.5). Secondly the rule introduces a new pair $(\mathfrak{l}|\rho_p)$. $\mathsf{sub}(\tilde{\pi}.\,\mathfrak{l}) = \bar{p}$ by side-condition of the rule and $\mathsf{sub}(\rho_p) = p$ by hypothesis, which completes the runnability proof. As the rule only replaces the $\rho$-component of a singly-anchored step and $\mathsf{sub}$ doesn't depend on such components, the subject of the resulting strategy is unchanged.

(7.6) As in the previous case the main event sequence is preserved by the transformation, and the complementarity of $\mathfrak{l}$ and $\rho_p$ is shown as in the previous case. As far as the subject is concerned, $\mathsf{sub}(\tilde{\pi} \lightning \rho) = \mathsf{sub}(\rho)$, and $\rho$ is the exact strategy prior to the transformation so we are done.

(7.7) The left hand side of the $\lightning$ symbol is runnable because both $\rho_p$ and $\tilde{\pi}.\,\mathfrak{l}$ are runnable, by hypothesis. The subject is preserved because the right hand side of $\lightning$ is the strategy prior to transformation.

(7.8) the $p_{\mathbf{R}}$ annotated dependency statement being consistent by hypothesis, $(\rho_p|\bullet).\,\phi$ is consistent, and therefore $(\rho_p|\bullet).\,\phi\,[\,s'\,]$ is runnable. Replacing that statement by $(\rho_p|\rho_0).\,\phi\,[\,s'\,]$ preserves runnability, as $\mathsf{sub}(\rho_0) = \bar{p}$ and $\mathsf{sub}(\rho_p) = p$. The subject of the strategy prior to transformation is $\mathsf{obj}(\rho_0)\,[\,s'\,]$. The subject of $\phi\,[\,s'\,]$ is $\mathsf{obj}(\rho_p)\,[\,s'\,]$, so the subject after transformation is

$$\mathsf{obj}(\rho_p)\,[\,r\,]\,\mathsf{subst}((\rho_p|\rho_0)) = \mathsf{obj}(\rho_p)\,[\,s'\,]\,\{{}^{\mathsf{obj}(\rho_0)}/_{\mathsf{obj}(\rho_p)}\} = \mathsf{obj}(\rho_0)\,[\,s'\,]$$

as required.

We now prove the second lemma, composition preserves consistency:

Following Definition 4.2.6 on page 35:

The first step simply combines into a single behavioural statement strategies from $\Gamma_1$ and $\Gamma_2$. As consistency of a statement is equivalent to runnability of all liveness strategies it contains, consistency of both $\Xi_{\mathrm{L}i}$ immediately implies consistency of $\Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2}$, *except* the observability requirement on $(\rho|\bullet)$-steps, as $\mathsf{sub}(\rho)$ being observable in one $\Gamma_i$ doesn't imply it being observable in $\Gamma_1 \odot \Gamma_2$. Note however that those strategies violating the observability requirement all have dependencies weaker or equal to $\mathsf{dep}_{\mathcal{K}}(\rho)$, by consistency of behavioural statements.

The second step performs a number of dependency reductions which, by Lemma 7.5.6, preserve consistency of the strategies, still disregarding the observability requirement.

Finally, the third step of Definition 4.2.6 removes statements depending on non-observable resources, thereby dropping all strategies that violated the observability requirement, so that $\Gamma_1 \odot \Gamma_2$ is consistent, now including the observability constraints.

### A.4.7 Closure Completes (Lemma 7.5.10)

Let $P$ and $\Gamma$ be as in the statement, let $\bigvee_{i\in I} \Phi_i$ be the local component of close $(\Gamma)$.

We show by induction on the size of a choice set $\tilde{\rho}$ that $\exists i \in I$ s.t. $\Phi_i$ doesn't contradict $\tilde{\rho}$, i.e. $\Phi$ is complete.

The base case ($\tilde{\rho} = \varnothing$) is immediate — if the choice set is empty it can't contradict any $\Phi_i$.

Fix a choice set $\tilde{\rho}$. Let $I_0 \subseteq I$ be the set of $i$ such that $\Phi_i$ doesn't contradict $\tilde{\rho}$. By induction hypothesis $I_0 \neq \varnothing$. Let $\rho_c$ be a selection strategy such that $\tilde{\rho} \cup \{\rho_c\}$ is a choice set according to Definition 7.3.4 (i.e. $\rho_c$ is runnable, doesn't contradict any $\rho \in \tilde{\rho}$ and all proper sub-strategies of $\rho_c$ are in $\tilde{\rho}$). We show that there is a non-empty subset $I_0' \subseteq I_0$ such that $j \in I_0'$ implies $\Phi_j$ doesn't contradict $\rho_c$.

Let $\hat{\imath} \in I_0$ be such that $\Phi_{\hat{\imath}}$ contradicts $\rho_c$, and specifically let $(s_k \triangleleft \varepsilon\colon \rho) \preceq \Phi_{\hat{\imath}}$ be such that $\rho$ contradicts $\rho_c$. If there is no such $\hat{\imath}$ then $I_0' = I_0$ and we're done.

As all sub-strategies of $\rho_c$ are in $\tilde{\rho}$ and $\hat{\imath} \in I_0$, $\Phi_{\hat{\imath}}$ doesn't contradict any sub-strategy of $\rho_c$.

Let $\rho_c = \tilde{\pi}_c.\,(\mathfrak{l}|\hat{\rho}_c)$. Following Definition 7.3.3 there is a sequence of steps $\tilde{\pi}.\,(\mathfrak{l}|\hat{\rho})$ contained in $\rho$ such that $\tilde{\pi}$ matches $\tilde{\pi}_c$, and $\hat{\rho}$ doesn't match $\hat{\rho}_c$. Let $q = \mathsf{sub}(\tilde{\pi}.\,\mathfrak{l})$. By runnability of $\rho$ (by hypothesis $\Gamma$ is consistent) and $\rho_c$, $\mathsf{sub}(\hat{\rho}) = \mathsf{sub}(\hat{\rho}_c) = \bar{q}$ (unless $\hat{\rho} = \bullet$ or $\hat{\rho}_c = \bullet$).

By pre-completeness of $\Phi$ (first point in Definition 7.5.9), $\bar{q}_{\mathbf{R}} \triangleleft \varepsilon_q\colon \rho_q.\,\phi_q \preceq \Phi_i$ where $\rho_q$ is a precursor of $\hat{\rho}_c$. Moreover (second point in Definition 7.5.9), there is a precursor $\rho_0$ of $\rho$ where $(\mathfrak{l}|\bullet)$ replaces $(\mathfrak{l}|\hat{\rho})$ and such that $(s_k \triangleleft \varepsilon_0\colon \rho_0) \preceq \Phi_i$.

Applying Definition 7.5.3, $\Phi_i \hookrightarrow \Phi_i \vee \Phi_i'$ where $\Phi_i'$ is obtained from $\Phi_i$ by repeatedly applying the transformations

- $\tilde{\pi}.\,(\mathfrak{l}|\bullet).\,\rho_2 \mapsto \tilde{\pi}.\,(\mathfrak{l}|\bullet) \lightning \tilde{\pi}.\,(\mathfrak{l}|\rho_q).\,\rho_2$ and

- $(\bullet|\tilde{\pi}.\,\mathfrak{l})\,[\,s'\,] \mapsto (\bullet|\tilde{\pi}.\,\mathfrak{l}) \lightning (\rho_q|\tilde{\pi}.\,\mathfrak{l}).\,\phi_q\,[\,s'\,]$.

As $\rho_q$ is a precursor of $\hat{\rho}_c$, $\Phi_i' \hookrightarrow \Phi_i' \vee \Phi_i''$ where $\Phi_i''$ is obtained from $\Phi_i'$ by further replacing $\rho_q$ by $\hat{\rho}_c$ in the rules above.

As $\Phi$ is closed, $\Phi \cong \Phi \vee \Phi_i' \vee \Phi_i''$, so there is $j \in I$ such that $\Phi_j \cong \Phi_i''$. As $\Phi_i$ doesn't contradict $\tilde{\rho}$ and $\Phi_j$ was obtained from $\Phi_i$ by moving sub-strategies around, $\Phi_j$ doesn't contradict $\tilde{\rho}$ either so we have $j \in I_0$. By construction $\Phi_j$ doesn't contradict $\rho_c$, so $j \in I_0'$ and therefore $I_0'$ can't be empty.

## A.4.8 Annotated Type System Soundness (Lemma 7.5.13)

The proof of the Lemma proceeds by induction on the proof sequence: Assuming for each rule that the typings in its assumptions are consistent and complete, we show that the typed process produced by the rule is consistent and complete as well. Rules (R-NIL) and (R-RES) are trivial, so we focus on (R-PAR), (R-SUM) and (R-PRE).

*Consistency of (*R-PAR*) strategies.* If a strategy is consistent in $P_i$ then it is also consistent in $P_1 \mid P_2$, so this case follows directly from Lemma 7.5.7

*Completeness of (*R-PAR*) strategies.* Assume both $\Gamma_i$ are complete and pre-complete for the corresponding $P_i$. Let $P = P_1 \mid P_2$. Let $\bigvee_{j \in J} \Phi_j$ and $\bigvee_{k \in K} \Phi_k$ respectively be the local behavioural statements of $\Gamma_1$ and $\Gamma_2$. As $\odot$ is a logical homomorphism, $\bigvee_{j \in J} \Phi_j \odot \bigvee_{k \in K} \Phi_k = \bigvee_{j \in J, k \in K}(\Phi_j \odot \Phi_k)$ is $\Gamma_1 \odot \Gamma_2$'s local behavioural statement before the closure operator is applied.

Let $\tilde{\rho}$ be a choice set. As both $\Gamma_i$ are pre-complete there are $j \in J$ and $k \in K$ such that both $\Phi_j$ and $\Phi_k$ are pre-complete with respect to $\tilde{\rho}$.

We show that the three points in Definition 7.5.9 are satisfied by $\Phi' = \Phi_j \odot \Phi_k$ with respect to $\tilde{\rho}$:

- As all liveness strategies in $\Phi'$ originate from $\Phi_j$ and $\Phi_k$ which are pre-complete (with respect to $\tilde{\rho}$) by hypothesis, strategies in $\Phi'$ don't self-contradict.

- Let $\rho = \pi_1 . \cdots . \mathfrak{l}_n$ be a strategy with $\mathsf{sub}_P(\rho) = p$. We construct a precursor $\rho'$ of $\rho$ such that $\mathsf{sub}_{P_i}(\rho') = p$ for some $i \in \{1, 2\}$.

  Reasoning by induction, for all $\rho_i \neq \bullet$ there is a $\rho'_i$ that is runnable in one of the $P_i$

  As $\rho$ is runnable $\mathfrak{l}_n$ must either be contained in one of $P_1$ and $P_2$. Assume it is in $P_1$, the proof for $P_2$ being identical but swapping all 1 and 2.

  Let $j < n$ be the largest number such that $\rho_j \neq \bullet$ and $\rho'_j$ is *not* runnable in $P_1$ (i.e. it is runnable in $P_2$). If there is no such $j$ we are done.

  Otherwise we give a procedure that transforms $\rho$ into a precursor $\hat{\rho}$ that is either $P_1$- or $P_2$-runnable, or that is such that $j$ strictly decreases. As $j$ must be positive and finite, applying this procedure a finite number of times will result in a $P_1$- or $P_2$-runnable precursor of $\rho$.

  If $\pi'_j$ substitutes $p' = \mathsf{sub}_{P_1}(\pi'_{j+1} . \cdots . \mathfrak{l}_n)$ by $p$ (i.e. there is $q$ such that $\mathsf{obj}_{P_1}(\mathfrak{l}_j)\,[\,q\,] = p'$ and $\mathsf{obj}_{P_2}(\rho'_j)\,[\,q\,] = p$) then set $\hat{\rho} = \rho'_j\,[\,q\,]$ which is, by hypothesis, $P_2$-runnable, so we're done.

  In all other cases, $\rho'_j$ is not used to compute $\mathsf{sub}_P(\rho)$ and it is safe to replace $\rho'_j$ by $\bullet$ to get $\hat{\rho}$.

  We now have a precursor $\rho'$ of $\rho$ that is $P_i$-runnable. So, as by hypothesis $\Phi_j$ (this is for $i = 1$, take $\Phi_k$ if $i = 2$) is pre-complete for $P_i$ with respect to $\tilde{\rho}$ and therefore contains a statement $p_{\mathbf{R}} \lhd \varepsilon : \rho_0 . \phi$ where $\rho_0$ is a precursor of $\rho'$ (and therefore of $\rho$ as well). By definition of the $\odot$ operator on behavioural statements, that exact same statement $p_{\mathbf{R}} \lhd \varepsilon : \rho_0 . \phi$ is contained in $\Phi'$ as well, as required.

- Assume $\Phi'$ contains an annotated liveness statement $p_k \lhd \varepsilon : \rho_2$ and let $\rho_1$ be a precursor of $\rho_2$. Then, applying $\odot$ backwards, one of $\Phi_i$ contains the same statement, and as it is pre-complete with respect to $\tilde{\rho}$, contains a statement $p_k \lhd \varepsilon' : \rho_0$ for some precursor of $\rho_0$. Applying $\odot$ back, the same statement $p_k \lhd \varepsilon' : \rho_0$ is contained in $\Phi'$, as required.

As this holds for any choice set $\tilde{\rho}$, $\bigvee_j \Phi_j \odot \bigvee_k \Phi_k$ is pre-complete. So, by Lemma 7.5.10 $\Gamma_1 \odot \Gamma_2$ is complete.

*Consistency of (*R-Pre*) strategies.* The typed process produced by this rule contains strategies in four places. The "$l$" strategy of local liveness is trivially runnable and has dependency $\top$. The local responsiveness strategy only contains strategies of the form $\bullet$ so it is trivially consistent as well. Strategies of remote behaviour are all of the form $(\bullet|l)\,[\,p\,]$ so they are runnable, and have dependency $\mathsf{dep}_{\mathcal{K}}(G) \wedge (\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}})$ which is equivalent to the declared $\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}}$. Finally, for the last factor (continuation $(l|\bullet).\,\Gamma \lhd \mathsf{dep}_{\mathcal{K}}(G)$), consider a dependency statement $s_k \lhd \varepsilon_0 : \rho$ in $\Gamma$. After prefixing and adding a dependency, it becomes $s_k \lhd (\varepsilon_0 \wedge \mathsf{dep}_{\mathcal{K}}(G)) : ((l|\bullet).\,\rho)$. $\bullet$-steps being always runnable, $\rho$'s runnability is preserved. Then (taking $P' = G^l.P$) $\mathsf{dep}_{P'}((l|\bullet).\,\rho) = \mathsf{dep}_{\mathcal{K}}(G) \wedge \mathsf{dep}_{P'}(\rho) = \mathsf{dep}_{\mathcal{K}}(G) \wedge \mathsf{dep}_{P}(\rho)$, so $\mathsf{dep}_{P}(\rho) \succeq \varepsilon_0$ (which holds as $\Gamma$ is consistent) implies $\mathsf{dep}_{P'}((l|\bullet).\,\rho) \succeq (\varepsilon_0 \wedge \mathsf{dep}_{\mathcal{K}}(G))$ which is what we needed. As all components are consistent, by Lemma 7.5.7, their composition also is.

*Completeness of (*R-Pre*) strategies.* As the composition of every type factor will perform a closure it is enough, by Lemma 7.5.10, to show that the type is pre-complete before the closure is performed. Every event in continuation is provided by the last factor. The subject of $G$ has a strategy provided by the local responsiveness factor, and its objects have responsiveness provided by the remote behaviour factor. However please see the note at the end of this section in case $G$ is replicated.

*Consistency of (*R-Sum*) strategies.* The strategies for the individual guards have length one and are therefore always runnable. The strategies in the components of the sum are assumed to be runnable by the premise $(\Sigma_i; \Phi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash'_{\mathcal{K}} {G_i}^{l_i}.P_i$ and the induction hypothesis.

*Completeness of (*R-Sum*) strategies.* Let $\tilde{\rho}$ be a choice set for the process $P = \sum_{i \in I} {G_i}^{l_i}.P_i$. By the non-contradiction condition, there must be $i \in I$ such that any $\rho \in \tilde{\rho}$ is either $l_i$ or of the form $(l_i|\bullet).\,\rho_0$, where $\rho_0$ is a selection strategy for $P_i$. Now assume some transition sequence $P \xrightarrow{\tilde{\mu}} P'$ does not contradict $\tilde{\rho}$. Because of the structure of $P$, the first transition in $\tilde{\mu}$ must be a labelled transition consuming one guard $G_{i'}$, which performs the choice $l_{i'}$. Since that transition does not contradict $\tilde{\rho}$, we must have $i = i'$, so completeness of the type for that transition sequence and the choice set $\tilde{\rho}$ follows, by the induction hypothesis, from completeness of $\Phi_{\mathrm{L}i}$ for the transition sequence ${G_i}^{l_i}.P_i \xrightarrow{\tilde{\mu}} P'$ and the choice set $\tilde{\rho}$.

# Appendix B

# Notation Index

The numbers between brackets indicate the page in which the item is first defined.

## B.1 Meta-variables

$a$, $b$, $c$, $d$ — channel names

$C$ — process context

$G$ — guards

$i$ — indexes

$I$ — indexing sets

$k$ — properties

$\mathcal{K}$ — set of properties

$l$ — events

$\mathfrak{l}$ — extended events

$m$ — multiplicities

$p$, $q$ — ports

$P$, $Q$ — processes

$\mathfrak{p}$ — extended ports

$s$ — branching

$x$, $y$, $z$ — more channel names

$\mathfrak{x}$, $\mathfrak{y}$ — extended names

$\alpha$, $\beta$, $\gamma$ — resources

$\Gamma$ — process types

$\delta$ — liveness strategy continuation

$\Delta$ — $\Gamma$, $\Xi$ or $\varepsilon$

$\varepsilon$ — dependencies

$\phi$ — responsiveness strategies

$\Phi$ — annotated behavioural statements

$\mu$ — transition labels

$\pi$ — liveness strategy step

$\rho$ — liveness strategies

$\sigma$ — channel types

$\xi$ — channel type behavioural statements (with parameter numbers instead of names)

$\Xi$ — behavioural statements

## B.2 Processes

$\mathbf{0}$ — idle process (14)
$\overline{a}\langle\tilde{x}\rangle$ — send $\tilde{x}$ over $a$ (14)
$a(\tilde{y})$ — receive something over $a$ and refer to it as $\tilde{y}$ (14)
$\overline{a}(\boldsymbol{\nu}\tilde{z})$ — private parameters, $(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{z}\rangle$ (14)
$G.P$ — run $G$ then $P$ (14)
$P\,|\,Q$ — parallel composition (14)
$P+Q$ — branching (14)
$\mathbf{!}\,G$ — replication (14)
$(\boldsymbol{\nu}x)\,P$ — $x$ is private in $P$ (14)
$?.P$ — may or may not run $P$ (63)
$\perp.P$ — will never run $P$ (61)
$\tau.P$ — runs $P$ after a $\tau$-transition (51)
$a \gg b$ — forwards $a$ to $b$ (33)
$P \oplus Q$ — internally selects $P$ or $Q$ (63)
$P \xrightarrow{\mu} P'$ — labelled transition (16)
$P \rightarrow P'$ — $\tau$-reduction $P \xrightarrow{\tau} P'$
$P \Rightarrow P'$ — weak transition (reflexive-transitive closure of $\rightarrow$)
$P \overset{\mu}{\Longrightarrow} P'$ — short for $P \Rightarrow\xrightarrow{\mu}\Rightarrow P'$

## B.3 Multiplicities

0, 1, $\star$, $\omega$ — zero, linear, plain, replicated (18)
$\#(G)$ — the multiplicity of $G$: 1 or $\omega$ (14)
$p^m$ — port $p$ has multiplicity $m$ (20)

## B.4 Resources

$\mathcal{U}$ — universal properties (31)
$\mathcal{E}$ — existential properties (31)
$\mathsf{proc}$ — the "process channel", connecting the process and its environment (94)
$\mathbf{A}$ — active (62)
$\mathbf{B}$ — bounded (118)
$\mathbf{D}$ — deterministic (96)
$\mathsf{proc}_{\mathbf{df}}$ — deadlock-free (102)
$\mathbf{I}$ — isolated (95)
$\mathbf{N}$ — never used in subject position (98)
$\mathbf{O}$ — used as output object (45)
$\mathsf{proc}_{\mathbf{ok}}$ — correct (all channels protocols are respected) (43)
$\mathbf{R}$ — responsive (45)
$\varpi$ — used at most a finite number of times (100)

## B.5 Types and Behavioural Statements

## B.6 Type Algebra

## B.7 Judgements

$\Gamma \vdash_{\mathcal{K}} P$ — $\Gamma$ types $P$, using elementary rules for all $k \in \mathcal{K}$ (56)

$\Gamma \vdash^{p}_{\mathcal{K}} P$ — $\Gamma$ types $P$, using elementary rules for all $k \in \mathcal{K}$, and $p$ as $P$'s parent port (94)

$\Gamma \models P$ — $\Gamma$ is semantically correct for $P$, following existential semantics (54)

$\Gamma \models_{\mathcal{U}} P$ — $\Gamma$ is semantically correct for $P$, following universal semantics (40)

$\Gamma \models_{\#} P$ — $\Gamma$ is semantically correct for $P$, following simple semantics (29)

$\Gamma \vdash'_{\mathcal{K}} P$ — annotated type $\Gamma$ types $P$, using elementary rules for all $k \in \mathcal{K}$ (91)

$\mathsf{good}_k(p \lhd \varepsilon, (\Gamma; P))$ — $p_k \lhd \varepsilon$ is immediately correct for $(\Gamma; P)$ (39)

$\mathsf{good}_{\#}(\Gamma; P)$ — $(\Gamma; P)$ is immediately correct with respect to the simple semantics (148)

$\mathsf{prop}_k(\sigma, G, m_i, m_o)$ — elementary guard rule for property $k$ (40)

$\mathsf{sum}_k(\tilde{p}, \Xi)$ — elementary sum rule for property $k$ (41)

## B.8 Annotated typed Processes

$G^l$ — annotated guard (73)

$(\mathfrak{l}|\rho)$ — doubly anchored liveness strategy step (74)

$(\mathfrak{l}|\rho]$ — singly anchored liveness strategy step (74)

$\bullet$ — communication partner in the environment (74)

$\rho\,[\,s\,]$ — parameter port(s) $s$ of $\rho$ (74)

$\mathfrak{l}.\,\rho$ — do step $\mathfrak{l}$ then proceed with $\rho$ (74)

$\tilde{\pi} \lightning \rho$ — try to follow $\tilde{\pi}$ but (at last step time) get hijacked and do $\rho$ instead. (74)

$\mathsf{wt}(\rho)$ — liveness strategy weight (88)

$\mathsf{sub}_P(\rho)$ — liveness strategy subject (79)

$\mathsf{obj}_P(\tilde{\pi})$ — liveness strategy objects (79)

$\mathsf{subst}_P(\tilde{\pi})$ — parameter substitution performed by sequence step (79)

$\mathsf{dep}^{-}_{\mathcal{K},P}(\rho)$ — $\rho$'s dependencies (81)

$\mathsf{rdep}_{\mathcal{K},P}(\tilde{\sigma}, \xi, \phi)$ — $\phi$'s dependencies as a responsiveness strategy for $(\tilde{\sigma}; \xi)$. (82)

$\sigma[\tilde{x}]_l$ — parameter instantiation for event $l$ (91)

$\gamma \lhd \varepsilon \colon \rho$ — $\rho$ is a liveness strategy for $\gamma$ (74)

$p_{\mathbf{R}} \lhd \varepsilon \colon \rho.\,\phi$ — $p$-guard $\rho$ has responsiveness strategy $\phi$ (77)

$\mathsf{mark}_\rho(P)$ — mark $P$ with $\rho$ (85)

$P \xrightarrow{\mu,(l_i|l_o)} P'$ — transition with corresponding strategy step (86)

$\mathrm{ran}(P)$, $\mathrm{ran}(\Gamma)$ — annotation removal (74, 77, 78)

# Index