Responsive Choice in Mobile Processes

Maxime Gamboni António Ravara

February 24, 2012

Abstract

We propose a general type notation, formal semantics and a sound, compositional, and decidable type system that characterise some liveness properties of distributed systems supporting choice. When such systems are specified using mobile processes, it is important to provide a number of liveness guarantees, such as, from a client's point of view, "If I send a request, will it eventually be received? Will it eventually be processed, and will I eventually obtain an answer?", or, from a server's point of view, "Will I eventually receive a request? Will my clients respect the protocol?".

We define two concepts, *activeness* (ability of sending/receiving on a channel) and *responsiveness* (ability to reliably conduct a conversation), that make the above properties precise, in particular what "eventually" and "respect the protocol" mean. The semantic definitions are respected by the type system, in the sense that the logical statements it outputs are correct descriptions of the process, according to the semantics.

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) or of object-oriented style programming (where a call matches one method out of a set). Our types and type system use *branching* and *selection* to capture activeness and responsiveness in process constructs necessary for such usage patterns.

Finally, *compositionality* features are offered through the use of *conditional properties* that permit analysing components of a system individually, and indicate, when applicable, what should be provided to the given process before the properties hold.

Keywords: Pi-calculus, Liveness Properties, Choice, Responsiveness, Type Systems.

1 Introduction

When describing a distributed or service-oriented system using mobile processes [MPW92, SW01], it is important to provide a number of liveness guarantees, such as, from a client's point of view, "If I send a request, will it eventually be received? Will it eventually be processed, and will I eventually obtain an answer?", or, from a server's point of view, "Will I eventually receive a request? Will my clients respect my communication protocol?". The work we present herein ensures these properties statically, allowing, e.g., to guarantee reliability of actual software or distributed protocols, or to prove validity of calculus encodings. The main contribution of this work is an integration of *choice* with

activeness and responsiveness, through a general type notation, formal semantics and a sound, compositional, and decidable type system. This work has three main ingredients:

First, *activeness* (ability to establish a connection) and *responsiveness* (ability to conduct a conversation for each connection) are liveness properties that have been studied, in more restricted forms, under the names of receptiveness [San99], lock-freedom [Kob02a] or responsiveness [AB08a]. Activeness is a generalisation of receptiveness both because communication is not required to succeed immediately but also because we may talk of output activeness, whereas receptiveness is only for inputs. Activeness of a channel end point (henceforth called *port*) is equivalent to lock-freedom of every instance of the complement port (including those in the environment). Acciai and Boreale's responsiveness is actually closer to what we call activeness than to our concept of responsiveness.

Secondly, *conditional properties* are statements of the form $\Delta \triangleleft \Theta$, where Δ and Θ are logical statements on channel activeness meaning that "Δ holds *provided* Θ is made available (e.g. through parallel composition)".

Thirdly, the language of processes, as well as the language of types, support the concepts of *selection* (or "internal choice") and *branching* (or "external choice"), abstract descriptions of *choices* made and communicated by processes.

One situation where choice and responsiveness appear together is Milner's encoding of Boolean values in the π-calculus [Mil93], which we'll use as running theme throughout this paper. Values are represented as receivers on two parameter channels: True replies to queries with a signal on the first parameter ($!\, b(tf).\bar{t}$) while False replies on the second one ($!\, b(tf).\bar{f}$). A Boolean is (input) active if it is able to receive a request, and (input) responsive if it is able to reply to all requests. Those two processes are instances of *selection* because they pick one behaviour out of a set of mutually exclusive permissions, by sending a signal to one parameter rather than to the other. A Random Boolean can be written $!\, b(tf).(\boldsymbol{\nu} x)\,(\bar{x} \mid (x.\bar{t}{+}x.\bar{f}))$, in which the selection is performed "at run-time" by the sum ("+"). A selection made by one process may cause *branching* in another process. Branching is typically implemented with the π-calculus sum operator. In the case of Booleans this occurs when *testing* a value: $\bar{b}(\boldsymbol{\nu} tf).(t.P{+}f.Q)$ runs P if b is True, and Q if b is False. The following process, which we'll use as running example throughout this paper, implements the "$r = a$ and b" logical circuit.

$$A = !\, r(tf).\overline{a}(\boldsymbol{\nu} t'f').(t'.\bar{b}\langle tf\rangle{+}f'.\bar{f}) \tag{1}$$

Upon receiving a request on r, process A first queries a. If it returns True (t') then the process returns on b the same channels received on r. If a returns False instead (f'), the process returns False ($\bar{f}$). 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). We shall use this process as a running example in the course of this paper. First by formally stating the property "r is responsive provided that both a and b are active and responsive" into a type, then we will prove that this statement is correct using semantic definitions, and finally, to illustrate our type system, we will show how to automatically infer that property from the process alone (and given that a, b and r are all Booleans).

The following example (in a π-calculus extended with numbers and a multiplication operator) implements a *multiplication service* that receives numbers and returns their product. At every step the client *selects* to send more numbers

("*more*") or request the result ("*done*"). Input (respectively, output) responsiveness of channel *prod* in this scenario means that the server (respectively, the client) will keep *progressing* until reaching a terminal state, i.e. until t is sent over r.

$$\text{Server} = \mathbf{!}\, prod(s).\overline{p_0}\langle s, 1\rangle \quad | \quad \mathbf{!}\, p_0(s,t).\bar{s}(\boldsymbol{\nu} more, done).\\ \big(more(s,n).\overline{p_0}\langle s, t \times n\rangle + done(r).\bar{r}\langle t\rangle\big)$$

$$\text{Client} = \overline{prod}(\boldsymbol{\nu} s).s(more, done).\overline{more}(\boldsymbol{\nu} s, 2).s(more, done).\overline{more}(\boldsymbol{\nu} s, 5).\\ s(more, done).\overline{done}(\boldsymbol{\nu} r).r(t).\overline{print}\langle t\rangle$$

To the best of our knowledge, no existing work is able to perform a static analysis of processes like (1). The usual approach for deciding whether names are active is to assign a single numerical level to name occurrences. But this does not allow for conditional properties, and moreover does not deal nicely with choice (specifically, with selection). In this case, when analysing r's continuation, as $\bar{t}$ may never get triggered (in case r returns False), it would require an infinite level, and similarly for $\bar{f}$. In other words, all a level-based system is able to say is "neither $\bar{t}$ nor $\bar{f}$ is guaranteed to ever be fired".

We need a typing system able to capture the fact that *exactly one* of $\bar{t}$ and $\bar{f}$ will eventually get triggered when r is queried, and, in contrast to level-based analysis, dependency-based systems as we have explored in the past [GR09] are naturally expanded with choice and branching operators, to express that sort of properties.

These three ingredients, responsiveness, choice and conditional properties, are put together into *behavioural statements*. Given a process and for every channel a *channel type* specifying its communication protocol, the type system constructs a *process type* containing a behavioural statement describing every property it was able to infer from the process (unless the process risks violating constraints such as linearity or arity of a channel, in which case it is rejected).

This paper uses an *incremental* presentation, starting with a basic arity checking system (Section 3) and then extending it step by step, with multiplicities (Section 4), choice (Section 5), and finally (Section 6) activeness and responsiveness. At each step we provide a type syntax and algebra, precise semantics and a type system.

A short version of this paper has previously been published as [GR10].

2 Processes

Let $\mathcal{N}$ be an at least countable set of *channel names* (often called just "names"), ranged over by a, b, c, d, r, x, y, z. Every channel x has two *ports*, its input (x) and output ($\bar{x}$) end points. Letter p ranges over ports. Symmetrically a port p has one channel given by $\mathrm{n}(p)$, defined as $\mathrm{n}(x) \stackrel{\text{def}}{=} x$, $\mathrm{n}(\bar{x}) \stackrel{\text{def}}{=} x$.

2.1 Syntax

Our target process calculus is the synchronous polyadic π-calculus with guarded sums and replication, with the grammar given in Table 1.

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

Table 1: Process Syntax

Tuples of channel names of length greater or equal to zero are denoted by $\tilde{x}$, $\tilde{y}$, or $\tilde{z}$. The arity of the tuple is the number of names in the sequence ($|\tilde{x}| = |x_1 \ldots x_n| = n$), and $\{\tilde{x}\}$ denotes the set of names in the sequence.

Free names $\mathrm{fn}(P)$ of a process P are defined as usual, binders being $(\boldsymbol{\nu} x)\, P$ (binding the free occurrences of x in P) and $a(\tilde{y}).P$ (binding the free occurrences of the names in $\tilde{y}$ in P).

A guard G has a *subject port* $\mathsf{sub}(G)$, a set of object names $\mathsf{obj}(G)$ and a set of *bound names* $\mathrm{bn}(G)$, given by:

- $\mathsf{sub}((\boldsymbol{\nu} \tilde{z})\, \overline{a}\langle \tilde{x} \rangle) \stackrel{\text{def}}{=} \overline{a}$ and $\mathsf{sub}(a(\tilde{y})) \stackrel{\text{def}}{=} a$
- $\mathsf{obj}((\boldsymbol{\nu} \tilde{z})\, \overline{a}\langle \tilde{x} \rangle) \stackrel{\text{def}}{=} \{\tilde{x}\}$ and $\mathsf{obj}(a(\tilde{y})) \stackrel{\text{def}}{=} \{\tilde{y}\}$
- $\mathrm{bn}((\boldsymbol{\nu} \tilde{z})\, \overline{a}\langle \tilde{x} \rangle) \stackrel{\text{def}}{=} \{\tilde{z}\}$ and $\mathrm{bn}(a(\tilde{y})) \stackrel{\text{def}}{=} \{\tilde{y}\}$.

Finally G has a *multiplicity* $\#(G)$ equal to ω if it is replicated, 1 otherwise.

2.2 Operational Semantics

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.

Definition 2.2.1 (Structural Congruence) Structural congruence $\equiv$ *is the smallest congruence on processes such that:*

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

Structural congruence is also helpful to give a succinct definition of top-levelness (Definition A.5.5 but also Lemma A.1.4).

Definition 2.2.2 (Transition Label) Transition labels *are given by*

$$\mu \;::=\; \tau \;\mid\; a(\tilde{x}) \;\mid\; (\boldsymbol{\nu} \tilde{z})\, \overline{a}\langle \tilde{x} \rangle$$

where $a \notin \tilde{z} \subseteq \tilde{x}$.

The labels in the above definition respectively stand for silent reduction, input and (bound) output. Whenever $|\tilde{z}| = 0$ we omit the binder $(\boldsymbol{\nu} \tilde{z})$.

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

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

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

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

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle} P' \quad Q \xrightarrow{a(\tilde{x})} Q' \quad \tilde{z} \cap \mathrm{fn}(Q) = \varnothing}{\begin{array}{l} P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{z})\,(P' \mid Q') \\ Q \mid P \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{z})\,(Q' \mid P') \end{array}}\ (\textsc{Com})$$

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

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

Table 2: Labelled Transition System

Definition 2.2.3 (Transition Relation) *The rules in table 2 inductively define a* labelled transition relation *on processes, written* $P \xrightarrow{\mu} P'$.

Note that $(\boldsymbol{\nu}b)\,\overline{a}\langle abab\rangle$ and $a(aa)$ are valid transition labels. The subject port $\mathsf{sub}(\mu)$ and object names $\mathsf{obj}(\mu)$ of a transition $\mu \neq \tau$ are defined similarly to those of a guard. Bound names $\mathrm{bn}(\mu)$ of μ are given by $\mathrm{bn}((\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle) = \{\tilde{z}\}$, and $\mathrm{bn}(\mu) = \varnothing$ for other cases. The set $\mathrm{n}(\mu)$ of names in a transition is defined as $\mathrm{n}(a(\tilde{x})) = \{\tilde{x}\} \cup \{a\}$, $\mathrm{n}((\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle) = \{\tilde{x}\} \cup \{a\}$ and $\mathrm{n}(\tau) = \varnothing$.

Let $\Rightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, and consider that $\xrightarrow{\tilde{\mu}}$ stands for $\xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n}$ for some $n \geq 0$.

3 Arity Type System

In order to expose the fundamental structure of our type algebra and type system we start with a very simple goal: a type system able to guarantee safety of a process against arity mismatches, a goal that was first achieved by Milner [Mil93]. In the next sections we will enrich the types and accompanying type system with multiplicities and behavioural information.

The basic idea for verifying a process P is safe is through a *channel type mapping* (ranged by Σ) that assigns to (at least) each free name a in P a *channel type* $\Sigma(a)$.

Our arity type system is *compositional* in the sense that correctness of a process $P_1|P_2$ with respect to a channel type mapping Σ is obtained by verifying P_1 and P_2 independently against that mapping, which requires channel types to contain enough information to describe the full behaviour of a process in term of arity. In particular, as names passed on channels can themselves be used as channels, it is necessary to have channel types include the types of their parameters even if the goal is only to check for arity mismatches.

Channel types are ranged over by σ and defined as follows.

Definition 3.0.4 (Channel Type — Arity) Channel types *are given by* $\sigma \ ::= \ \langle \sigma_1, \ldots, \sigma_n \rangle$ *with* $n \geq 0$. *Let the* arity *of a channel type be given by* $|\, \langle \sigma_1, \ldots, \sigma_n \rangle \,| = n$.

Recursion ends at the parameter-less channel type $\lambda \stackrel{\text{def}}{=} \langle\rangle$.

Definition 3.0.5 (Channel Type Mapping) *A* channel type mapping*, denoted* Σ*, is a function associating channel types to names. We use the notation* $\Sigma = \{a_1 : \sigma_1, a_2 : \sigma_2, \ldots\}$ *or* $\{\tilde{a} : \tilde{\sigma}\}$ *when defining such a mapping directly and write* $\varnothing$ *for the empty mapping.*

For instance the two processes $a(x).\overline{x}\langle b \rangle$ and $a(x).\overline{x}\langle bc \rangle$ respectively treat a as having the monadic types $\langle\, \langle \sigma_b \rangle \,\rangle$ and $\langle\, \langle \sigma_b, \sigma_c \rangle \,\rangle$, σ_b and σ_c being types for b and c. Note the double pair of brackets, the inner channel types $\langle \sigma_b \rangle$ and $\langle \sigma_b, \sigma_c \rangle$ being that of the bound name x.

Channel type mappings can't give the type of bound names in a process, so we insert them in the process itself, by annotating binders. The P- and T- rules from Table 1 are modified as follows:

$P \ ::= \ (P|P) \ \mid \ (\boldsymbol{\nu} x : \sigma)\, P \ \mid \ S \ \mid \ \mathbf{0}$

$T \ ::= \ (\boldsymbol{\nu} x : \sigma)\, T \ \mid \ a(\tilde{y}) \ \mid \ \overline{a}\langle \tilde{x} \rangle$

where σ indicates x's type in P and T. We sometimes write $(\boldsymbol{\nu} \tilde{x} : \tilde{\sigma})\, P$ or even $(\boldsymbol{\nu}\Sigma)\, P$ for $(\boldsymbol{\nu} x_1 : \sigma_1) \ldots (\boldsymbol{\nu} x_n : \sigma_n)\, P$, and in examples we omit the channel type annotations when they are nor relevant. Input parameters need not be similarly annotated, as their types can be directly obtained from the type of the subject that carries them. Output transition labels are similarly modified and are written $(\boldsymbol{\nu} \tilde{z} : \tilde{\sigma})\, \overline{a}\langle \tilde{x} \rangle$ where $|\tilde{z}| = |\tilde{\sigma}|$, and the transition rules (Open), (New) and (Com) are modified the obvious way by carrying the channel types around. For instance (Open) works on type-annotated process as follows:

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

In process (1), the (bound) channels t, f, t' and f' have type λ, while r, a and b have the Boolean channel type $\mathsf{Bool} \stackrel{\text{def}}{=} \langle \lambda, \lambda \rangle$. This is formalised with the following channel type mapping

$$\Sigma = \{a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}\} \tag{2}$$

3.1 Semantics

Before proposing a type system we recall the property the type system has to guarantee and define correctness of a channel type mapping for a process. This definition is along the lines of the one used in Igarashi and Kobayashi's Generic Type System [IK01], and more permissive than that proposed in Milner's Sorting [Mil93] as it permits a given name to change arity as the process evolves, as in $(\boldsymbol{\nu} a)\left(a(x).\overline{a}\langle bc\rangle.P \mid \overline{a}\langle x\rangle.a(de).Q\right)$ (where a is first used to exchange the unary message $\langle x\rangle$ and then the binary message $\langle bc\rangle$, yet exhibits no arity mismatch because no binary prefix can become available before the unary prefixes are consumed). Our type systems will however enforce a fixed arity for each name, therefore rejecting the above process.

We will then prove that existence of a correct type mapping is a sufficient (but not necessary) condition for safety against arity mismatches.

Definition 3.1.1 (Arity Mismatch) *A process P is said* in error with respect to arity *(written $\mathsf{bad}_{\mathsf{arity}}(P)$) if $P \equiv (\boldsymbol{\nu}\tilde{z} : \tilde{\sigma})\left((a(\tilde{x}).Q \mid \overline{a}\langle\tilde{y}\rangle.Q') \mid Q''\right)$ for some a, Q, Q', Q'', $\tilde{x}$, $\tilde{y}$, $\tilde{z}$ and $\tilde{\sigma}$, where $\tilde{x} \neq \tilde{y}$.*

For instance $\bar{a} \mid a(y).P$ is in error with respect to arity, but none of $\bar{a} \mid a.P$, $\bar{a} \mid \overline{a}\langle x\rangle.P$, $\bar{a} + a(y).P$ and

$$(\boldsymbol{\nu} x : \lambda)\left(\bar{x} \mid x.(\bar{a} \mid a(y).P_0)\right) \tag{3}$$

are. The first three do not attempt to make two terms communicate with differing arities, and the last one does not attempt to do so *immediately*, but will be in error after one τ-reduction, prompting for the following more useful definition.

Definition 3.1.2 (Arity τ-Safety) *A process P is said* τ-safe *against arity mismatches, written $\models^{\tau}_{\mathsf{arity}} P$, if there is no process Q such that $P \Rightarrow Q$ and $\mathsf{bad}_{\mathsf{arity}}(Q)$.*

Assuming process P being the one given in (3), it should be clear that $\not\models^{\tau}_{\mathsf{arity}} P$, as $P \xrightarrow{\tau} Q = (\boldsymbol{\nu} x : \lambda)\left(\bar{a} \mid a(y).P_0\right)$ and $\mathsf{bad}_{\mathsf{arity}}(Q)$.

The "τ" annotation is used because we only permit τ-reductions from P to Q. This restriction is required because requiring safety after arbitrary labelled transitions would mark most non-trivial processes unsafe, because mismatches can be caused by the transitions themselves, as in the following example:

$$P = a(x).\overline{x}\langle b\rangle \mid c(yz).Q \xrightarrow{a(c)} \overline{c}\langle b\rangle \mid c(yz).Q = P' \tag{4}$$

where P' is in error but P is safe. However restricting ourselves to τ-safety would be unfortunate as it amounts to treating a process as a closed system that doesn't interact with its environment, going against a fundamental feature of the π-calculus, the parallel composition operator $\mid$ that allows putting together independently written components. In the following section we discuss an intermediate approach in the form of a *typed* transition relation.

3.2 Dynamic Types and the Transition Operator

We describe in this section a *transition operator* on types to answer the following question: If a process P has a channel type mapping Σ, and $P \xrightarrow{\mu} P'$, what is the channel type mapping for P'? The transition operator $\wr$ applies the transition label μ to Σ and returns $\Sigma \wr \mu$ as a channel type mapping for P', or is undefined whenever μ should not be permitted.

Taking a closer look at (4), one can note that the channel type mapping before the transition must be

$$\Sigma = \{a : \langle\, \langle \sigma_b \rangle \,\rangle, c : \langle \sigma_y, \sigma_z \rangle\} \tag{5}$$

for some σ_b, σ_y and σ_z. A transition $a(c)$ then amounts to attempt unifying a's parameter type $\langle \sigma_b \rangle$ with $\Sigma(c)$ which of course can't be done, due to the differing number of parameters. This reasoning is formally carried out using the following:

Definition 3.2.1 (Parameter Instantiation) *Let $\sigma = \langle \tilde{\sigma} \rangle$ be an n-adic channel type and $\tilde{x}$ a sequence of n names such that $\forall i, j$: $x_i = x_j$ implies $\sigma_i = \sigma_j$. Then* instantiating *σ with $\tilde{x}$, written $\sigma[\tilde{x}]$, yields the channel type mapping $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$.*

In the above scenario we have $\sigma_a[c] = \{c : \langle \sigma_b \rangle\}$.

Note that the definition required special care in case two parameters may be the same name, to prevent a single name to be mapped to two different types. As similar situations will happen often we introduce the following operator:

Definition 3.2.2 (Channel Type Mapping Conjunction Operator) *The* conjunction *of two channel type mappings Σ_1 and Σ_2, denoted $\Sigma_1 \wedge \Sigma_2$, is well-defined if and only if, for all x, $\Sigma_1(x) = \sigma_1$ and $\Sigma_2(x) = \sigma_2$ implies $\sigma_1 = \sigma_2$. We then have $\Sigma_1 \wedge \Sigma_2 \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$.*

We may now rewrite the definition for $\sigma[\tilde{x}]$ as follows:

$$\langle \sigma_1 \ldots \sigma_n \rangle [x_1 \ldots x_n] \stackrel{\text{def}}{=} \{x_1 : \sigma_1\} \wedge \cdots \wedge \{x_n : \sigma_n\} \tag{6}$$

We are now equipped to give a definition for the transition operator:

Definition 3.2.3 (Transition Operator — Arity) *Let Σ be a channel type mapping and μ a transition label. Then the* transition operator *$\wr$ yields, for Σ and μ, the channel type mapping $\Sigma \wr \mu$ defined as follows:*

- $\Sigma \wr \tau \stackrel{\text{def}}{=} \Sigma$
- $\Sigma \wr a(\tilde{x}) \stackrel{\text{def}}{=} \Sigma \wedge (\Sigma(a))[\tilde{x}]$
- $\Sigma \wr (\boldsymbol{\nu} \tilde{z} : \tilde{\sigma})\, \overline{a}\langle \tilde{x} \rangle \stackrel{\text{def}}{=} \Sigma \wedge (\Sigma(a))[\tilde{x}] \wedge \tilde{z} : \tilde{\sigma}$

Whenever any of the involved operators isn't well-defined we say the entire transition operator isn't defined.

Let us apply this operator to Σ given in (5) and the transition (4), recalled below.

$$P = a(x).\overline{x}\langle b\rangle \mid c(yz).Q \xrightarrow{a(c)} \overline{c}\langle b\rangle \mid c(yz).Q = P'$$

By definition, $\Sigma \wr a(c) = \Sigma \wedge (\Sigma(a))[c] = \Sigma \wedge \langle\, \langle \sigma_b \rangle \,\rangle[c] = \Sigma \wedge \{c : \langle \sigma_b \rangle\}$, which is rejected by $\wedge$'s definition, as $\Sigma(c) = \langle \sigma_y, \sigma_z \rangle \neq \langle \sigma_b \rangle$.

Another example, the transition $A \xrightarrow{r(uv)} A'$ where A is given by (1) is modelled by $\Sigma' = \Sigma \wr r(uv)$ where Σ is given by (2). The reader may check that applying Definition 3.2.3 results in $\Sigma' = \{a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}, u : \lambda, v : \lambda\}$ as a type for A'.

We conclude this section with the following definition, that connects transitions on type mappings and transitions on processes:

Definition 3.2.4 (Transition on Typed Processes) $(\Sigma; P) \xrightarrow{\mu} (\Sigma'; P')$ *if* $P \xrightarrow{\mu} P'$ *and* $\Sigma \wr \mu$ *is well-defined and equal to* Σ'.

A pair $(\Sigma; P)$ is called a *typed process*.

3.3 Semantics — Channel Type Mapping

We now provide semantics of channel type mappings by way of a generalisation of Definition 3.1.1.

Definition 3.3.1 (Typed Arity Mismatch) *A typed process* $(\Sigma; P)$ *is said* in error *(written* $\mathsf{bad}_{\mathsf{arity}}(\Sigma; P)$*) if* $P \equiv (\boldsymbol{\nu}\Sigma')\,(G.Q \mid R)$ *for some* Σ', G, Q, R, *with* $\mathsf{sub}(G) = a$ *and* $|\mathsf{obj}(G)| \neq |\sigma|$ *where* $\sigma = \Sigma'(a)$ *if* $a \in \mathrm{dom}(\Sigma')$ *or* $\sigma = \Sigma(a)$ *if* $a \notin \mathrm{dom}(\Sigma')$.

If two top-level guards in a process disagree on a channel's arity, no channel type mapping can agree with both:

Lemma 3.3.2 *Let* $(\Sigma; P)$ *be a typed process.*
Then $\mathsf{bad}_{\mathsf{arity}} P$ *implies* $\mathsf{bad}_{\mathsf{arity}}(\Sigma; P)$.

We discussed in the previous section how the transition operator rules out transitions that could not be caused by an environment conforming to the channel types given by Σ.

We may now generalise Definition 3.1.2 to use arbitrary transition labels and not just τ-reductions, thereby addressing the concerns given at the time.

Definition 3.3.3 (Arity Safety) *A typed process* $(\Sigma; P)$ *is* safe *against arity mismatches (written* $\Sigma \models_{\mathsf{arity}} P$*) if*

- *For any sequence* $P \xrightarrow{\tilde{\mu}} P'$ *where all bound output objects in* $\tilde{\mu}$ *are fresh*[1] *and do not appear in any other transition, and all input objects are fresh and distinct,* $\Sigma \wr \tilde{\mu}$ *is well defined, and*
- *there is no transition sequence* $(\Sigma; P) \xrightarrow{\tilde{\mu}} (\Sigma'; P')$ *such that* $\mathsf{bad}_{\mathsf{arity}}(\Sigma'; P')$.

[1] A name ocurs fresh in a process P if it does not appear in any type or subprocess of P.

$$\frac{-}{\Sigma \vdash_{\mathsf{arity}} \mathbf{0}} \text{ (E-Nil)} \qquad \frac{\Sigma \vdash_{\mathsf{arity}} P \qquad \Sigma(x) = \sigma}{(\boldsymbol{\nu} x)\,\Sigma \vdash_{\mathsf{arity}} (\boldsymbol{\nu} x : \sigma)\,P} \text{ (E-Res)}$$

$$\frac{\forall i : \Sigma_i \vdash_{\mathsf{arity}} P_i}{\Sigma_1 \wedge \Sigma_2 \vdash_{\mathsf{arity}} P_1 \mid P_2} \text{ (E-Par)} \qquad \frac{\forall i : \Sigma_i \vdash_{\mathsf{arity}} S_i}{\Sigma_1 \wedge \Sigma_2 \vdash_{\mathsf{arity}} S_1 + S_2} \text{ (E-Sum)}$$

$$\frac{\Sigma \vdash_{\mathsf{arity}} P}{\{\mathsf{sub}(G) : \sigma\} \wedge (\boldsymbol{\nu}\mathrm{bn}(G)) \Big(\Sigma \wedge \ \sigma[\mathsf{obj}(G)] \Big) \ \vdash_{\mathsf{arity}} G.P} \text{ (E-Pre)}$$

Table 3: Arity Type System Rules

For example, $(\Sigma; P)$ given in (4) and (5) is safe against arity mismatches (assuming Q is), because setting a's object to a fresh name we get $\Sigma \wr a(d) = \Sigma \wedge \{d : \langle \sigma_b \rangle\} = \Sigma \cup \{d : \langle \sigma_b \rangle\}$. On the other hand $a(b).(b(x) \mid \overline{b}\langle yz \rangle)$ is not safe because even with a fresh parameter an input on a leads to a process in error. Note that the latter process is (vacuously) τ-safe, however.

The following lemma is easily shown by contraposition, as $P \xrightarrow{\tau} Q$ implies $(\Sigma; P) \xrightarrow{\tau} (\Sigma; Q)$ for any Σ.

Lemma 3.3.4 *Let $(\Sigma; P)$ be a typed process.*
Then $\Sigma \models_{\mathsf{arity}} P$ implies $\models^{\tau}_{\mathsf{arity}} P$.

3.4 Type System

We now propose a type system that provides a sound and decidable characterisation of safety from arity mismatches.

Definition 3.4.1 (Arity Type System) Typability *of a typed process $(\Sigma; P)$ with respect to* arity*, written $\Sigma \vdash_{\mathsf{arity}} P$, is inductively given by the rules in Table 3.*

All operators covered in the above rules have been covered before, with the exception of *restriction*, used in (E-Res) and (E-Pre), that mirrors the corresponding process constructor by dropping the type of the channel being bound.

Definition 3.4.2 (Restriction of a Channel Type Mapping) *Let Σ be a channel type mapping and x a name contained in* $\mathrm{dom}(\Sigma)$*. Then $(\boldsymbol{\nu} x)\,\Sigma \stackrel{\text{def}}{=} \Sigma \setminus \{x : \Sigma(x)\}$ (where $\setminus$ is the set subtraction operator).*

Properties. Like all type systems covered in this paper, the arity type system enjoys a number of properties. We only state them here without proof, as they are consequences of the corresponding properties of the most general type system given in Section 6.

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

Proposition 3.4.3 (Subject Congruence) *If $\Sigma \vdash_{\mathsf{arity}} P \equiv P'$ then $\Sigma \vdash_{\mathsf{arity}} P'$.*

This is a consequence of Proposition 7.3.1 on page 51.

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

Proposition 3.4.4 (Subject Reduction) *Let* $\Sigma \vdash_{\mathsf{arity}} P$ *and* $(\Sigma; P) \xrightarrow{\mu} (\Sigma'; P')$. *Then* $\Sigma' \vdash_{\mathsf{arity}} P'$

This is a consequence of Proposition 7.3.2, proved in Section A.3.

The arity type system is sound in the following sense:

Proposition 3.4.5 (Arity Type System Soundness) *For any typed process* $(\Sigma; P)$, $\Sigma \vdash_{\mathsf{arity}} P$ *implies* $\Sigma \models_{\mathsf{arity}} P$.

This is a consequence of Proposition 7.3.4, proved in Section A.5.

4 Multiplicities

In this section we enrich the type grammar, type algebra and type system to incorporate *multiplicities*. Consider the following scenario:

A process A wants to transmit a value v to a process B, which then sends a reference to the same value to process C. Rather than transmitting the full data in the first message, A actually creates a process $[\![v]\!]_u$ encoding the value v and providing access to its content through channel u, and sends the name u to B. B then sends the same name u to C:

$$A \mid B \mid C = \overline{b}(\boldsymbol{\nu} u).[\![v]\!]_u \mid b(x).\overline{c}\langle x\rangle \mid c(y).\ldots$$

Both B and C, to decode the value, can send a message on u, which is then replied to by A. However C now has the potential to change the value v as it appears to B, by creating another receiver on u:

$$C = c(y).\mathbf{!}\, y(\ldots).\ldots$$

Now, if the scheduler is fair, on average one half of the decoding requests sent by B will actually be intercepted by C. To avoid this, the process B would have to send a new channel u' to C and run a forwarder $u' \gg u$.[2] Another way which we are going to explore in this section is to constrain the *multiplicities* [KPT99, San99] of channel u.

Multiplicities, 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 are used once 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 $\mathbf{!}\, a(x).\bar{x}$) or not.

The issue in the above scenario 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.

That scenario involves the following multiplicities:

[2] That resends all message received on u to u': $\mathbf{!}\, u(z).\overline{u'}\langle z\rangle$.

1. *Uniform* [San99] or ω 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 *affine* [HY02], meaning that it may occur at most 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 encoding $\bar{a} \mid a.P \mid a.Q$ 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, ω and $\star$, standing respectively for "at most one non-replicated use", "at most one replicated use" and "unconstrained".[3] We will also need a multiplicity 0 for ports that must not be used at all.

4.1 Channel Types

Before proceeding further let us generalise channel types (Definition 3.0.4) to include multiplicities. We choose a notation that ensures channel types remain constant over time (are preserved by transition) and space (preserved by composition). However neither of these properties hold for multiplicities.

For instance (a being assumed linear, and separating multiplicities of the process and those of its environment with the symbol "$\blacktriangleleft$"), in $a|b \xrightarrow{a} b$, a's multiplicities evolve as $a^1, \bar{a}^0 \blacktriangleleft a^0, \bar{a}^1 \rightarrow a^0, \bar{a}^0 \blacktriangleleft a^0, \bar{a}^0$, and in $a|\bar{a} \xrightarrow{\tau} \mathbf{0}$, they evolve as $a^1, \bar{a}^1 \blacktriangleleft a^0, \bar{a}^0 \rightarrow a^0, \bar{a}^0 \blacktriangleleft a^0, \bar{a}^0$.

Multiplicities aren't preserved by composition either. For instance, in $P = (a \mid b \mid \bar{a})$, a has multiplicities $a^1, \bar{a}^0 \blacktriangleleft a^0, \bar{a}^1$ in the first component, $a^0, \bar{a}^0 \blacktriangleleft a^1, \bar{a}^1$ in the second, and $a^0, \bar{a}^1 \blacktriangleleft a^1, \bar{a}^0$ in the third. In P, a has multiplicities $a^1, \bar{a}^1 \blacktriangleleft a^0, \bar{a}^0$. So, in a single process, a single channel has four different sets of multiplicities.

All these considerations suggest separating channel types and channel multiplicities, so that a channel type contains multiplicity information for its parameters, but not its own multiplicities. We write p^m to express the fact that port p has multiplicity m, and refer to parameter channels by their number, starting from 1. For instance $\bar{2}^\omega$ states that the output port of the channel's second parameter must appear at most once, and that single occurrence must be replicated.

All examples we have considered so far have been *input-output-alternating*, in that input processes only output on their parameters, and an output $\overline{a}\langle b\rangle$ may only prefix or be composed with *inputs* at b. If that were always the case we could write a channel type as a pair

$$\sigma \quad ::= \quad \left\langle \tilde{\sigma}; \tilde{p}^{\tilde{m}} \right\rangle \tag{7}$$

[3] Lower bounds to express multiplicities like "At least once" or "Exactly once" are obtained using *activeness* as shown later.

where the first component gives the types of the parameters and the second component associates multiplicities m_i to parameter ports p_i.

However, not all process satisfy that alternation property, and (7) is not sufficient to accurately describe such cases. For instance a "server creator" $!\,a(x).!\,x(y).Q$ creates a one parameter server with body Q on all names sent to it. In that example, a's input uses its parameter's input capabilities and therefore a's type is not alternating. Following (7) we'd get something like $\langle\sigma; 1^\omega, \bar{1}^\star\rangle$ as a type for a, but that type doesn't express the fact that 1^ω should be provided by a's input and not its output: exactly the same type would be given for $\overline{a}\langle b\rangle.!\,b(y).Q$ where the input on x is provided by the *output* of a, and yet composing these two processes no longer respects the channel type.

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. We arrive at the following final notation for channel types with multiplicities:

$$\sigma \;::=\; \langle\tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}}\rangle \tag{8}$$

where ξ_{I} is a set of parameter multiplicities that the channel's input receivers may use, and ξ_{O} those that the output prefixes may use.

To explicitly write such a set of multiplicities we use the notation ${p_1}^{m_1} \wedge \ldots \wedge {p_n}^{m_n}$, where all p_i must be pairwise disjoint and "$\wedge$" is read "and". An empty set is denoted $\top$. We chose this notation inspired from propositional logic to have it easily extended in Section 5 with other logic connectives. This separation of parameter multiplicities into ξ_{I} and ξ_{O} is also 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.

For instance, consider a two linear parameter channel a, whose first parameter is alternating and second is not:

$$a(xy).(\bar{x}|y) \,|\, \overline{a}\langle bc\rangle.(b|\bar{c}) \tag{9}$$

The input on a uses the first parameter (denoted 1, appearing as b and x in (9)) with multiplicities $1^0 \wedge \bar{1}^1$, and the a-output uses that first parameter with multiplicities $1^1 \wedge \bar{1}^0$. The second parameter (2, c and y) has multiplicities $2^1 \wedge \bar{2}^0$ on a's input and $2^0 \wedge \bar{2}^1$ on a's output, which is all packed together into the following channel type for a:

$$a: \quad \big\langle \lambda, \lambda \quad ; \quad 1^0 \wedge \bar{1}^1 \wedge 2^1 \wedge \bar{2}^0 \quad ; \quad 1^1 \wedge \bar{1}^0 \wedge 2^0 \wedge \bar{2}^1 \big\rangle$$

Note that a's multiplicities appear nowhere in that statement.

Even though in this example the parameter multiplicities look very symmetric they need not be so. For instance the type $\big\langle \lambda; 1^0 \wedge \bar{1}^\star; 1^\star \wedge \bar{1}^\star \big\rangle$ is for a channel whose input side may only use the parameter in output position, but whose output may use the parameter without restrictions. Also note that even though ω is most commonly associated with channel inputs it does not need to be. For instance when a channel is used as a "value provider" where one process is responsible for providing a constant or the address of some service ("$v := 42$" is encoded as $!\,\overline{v}\langle 42\rangle$) and any process interested in knowing the constant value or the service address may do a v-input ($v(x).P$ where P may use that value or address as x), it is expected that one v-output always be available ($\overline{v}^\omega$), while inputs are not restricted ($v^\star$).

4.2 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 τ-reductions following $\overline{a}(\boldsymbol{\nu}\tilde{z}).E \,|\, a(\tilde{z}).P \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{z})\,(E \,|\, P)$.

We adapt the previously introduced channel type notation (8) for process types. Parameters 1, 2 etc become pairwise distinct channel names z_i free in the process being abstracted and we replace the second semi-colon by "◀" to explicitly mark the process/environment interface. This gives the following notation for a process type, ranged over by Γ:

$$(\Sigma \quad ; \quad \Xi_{\mathrm{L}} \quad \blacktriangleleft \quad \Xi_{\mathrm{E}}) \tag{10}$$

where Ξ_{L} and Ξ_{E}, both of the form $\bigwedge_{z\in\mathrm{dom}(\Sigma)} \left({z_i}^{m_i} \wedge \overline{z_i}^{m_i'}\right)$, respectively tell how the channel usages are divided between *local* (P) and *environment* (E).

Consider for example the process $P = \mathbf{!}\, u(x).\bar{x}$. Wrapping it into an input as described above gives $a(u).P$. In that process, the channel type for a is

$$\left\langle \left\langle \lambda; 1^0 \wedge \bar{1}^1; 1^1 \wedge \bar{1}^0 \right\rangle; 1^\omega \wedge \bar{1}^0; 1^0 \wedge \bar{1}^\star \right\rangle$$

which is transformed into a process type for P as follows:

$$\left(u : \left\langle \lambda; 1^0 \wedge \bar{1}^1; 1^1 \wedge \bar{1}^0 \right\rangle; u^\omega \wedge \bar{u}^0 \blacktriangleleft u^0 \wedge \bar{u}^\star\right)$$

Now consider a process $E = \overline{u}\langle t\rangle.t$ acting as the environment for P. The interaction $P \xrightarrow{u(t)} P|\bar{t} \xrightarrow{\bar{t}} P$ with that process corresponds to the reduction $a(u).P \,|\, \overline{a}(\boldsymbol{\nu}u).E \to (\boldsymbol{\nu}u)\left(\left(\mathbf{!}\, u(x).\bar{x}\right) \,|\, \left(\overline{u}\langle t\rangle.t\right)\right) \to (\boldsymbol{\nu}u)\left((P|\bar{t}) \,|\, t\right) \to (\boldsymbol{\nu}u)\left(P \,|\, \mathbf{0}\right)$. The process type for the intermediary form $P|\bar{t}$ would be

$$\left(u : \sigma_u, t : \lambda \;;\; t^0 \wedge \bar{t}^1 \;\blacktriangleleft\; t^1 \wedge \bar{t}^0\right)$$

where σ_u is u's type seen before.

Finally, after t has been consumed, we get

$$\left(u : \sigma_u, t : \lambda \;;\; t^0 \wedge \bar{t}^0 \;\blacktriangleleft\; t^0 \wedge \bar{t}^0\right)$$

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

$$\left(\ldots, t : \lambda \;;\; \ldots \wedge t^0 \wedge \bar{t}^0 \;\blacktriangleleft\; \ldots \wedge t^1 \wedge \bar{t}^1\right)$$

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 can now be seen as E passing t's output capability to P.

$$\left(t : \lambda; t^0 \wedge \bar{t}^0 \blacktriangleleft t^1 \wedge \bar{t}^1\right) \to \left(t : \lambda; t^0 \wedge \bar{t}^1 \blacktriangleleft t^1 \wedge \bar{t}^0\right)$$

The type of process (1) with multiplicity information is given by

$$\Gamma = \big(a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}; \\ r^\omega \wedge \bar{r}^0 \wedge a^0 \wedge \bar{a}^\star \wedge b^0 \wedge \bar{b}^\star \blacktriangleleft r^0 \wedge \bar{r}^\star \wedge a^\omega \wedge \bar{a}^\star \wedge b^\omega \wedge \bar{b}^\star\big) \quad (11)$$

where

$$\mathsf{Bool} \stackrel{\text{def}}{=} \langle \lambda, \lambda; 1^0 \wedge \bar{1}^1 \wedge 2^0 \wedge \bar{2}^1; 1^1 \wedge \bar{1}^0 \wedge 2^1 \wedge \bar{2}^0 \rangle$$

As exhaustive process types can become rather verbose, we shall from now on use the following simplifying conventions:

Convention 4.2.1 (Notation for Multiplicities)

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. *Exponents equal to one may be omitted.*
3. *In addition, the local component* Ξ_{L} *of a process type with channel type mapping* Σ *should be understood as follows:*

$$\Xi_{\mathrm{L}} \wedge \bigwedge_{x \in \mathcal{N} \setminus \mathrm{dom}(\Sigma)} \left(x^0 \wedge \bar{x}^0\right)$$

For instance we write $\bar{a}$ instead of $\bar{a}^1$.

The empty, or neutral, type $(\varnothing; \top \blacktriangleleft \top)$ describes the idle process $\mathbf{0}$, or any process with no free names, by analogy with the parameter-less channel λ. No channels a need be mentioned, as all have 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.

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.

In that simpler notation, (11) may be written

$$\Gamma = \big(a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}; r^\omega \wedge \bar{a}^\star \wedge \bar{b}^\star \blacktriangleleft r^0 \wedge a^\omega \wedge b^\omega\big) \quad (12)$$

where

$$\mathsf{Bool} \stackrel{\text{def}}{=} \langle \lambda, \lambda; \bar{1} \wedge \bar{2}; 1 \wedge 2 \rangle$$

4.3 Algebra

We now extend the operators and relations seen in the previous section for channel type mappings to work with process types.

We start with a *weakening* relation. A type is *weaker* than another one if it describes at least all processes described by the latter (and strictly weaker if it describes processes the latter doesn't).

Definition 4.3.1 (Weakening — Multiplicities) *The* weakening *relation* $\preceq$ *on process types is the least reflexive and transitive relation such that:*

- *For all* p *and* m, $p^0 \preceq p^m \preceq p^\star$

- $\bigwedge_{i\in I} {p_i}^{m_i} \preceq \bigwedge_{i\in I} {p_i}^{m'_i}$ *iff* $\forall i \in I : {p_i}^{m_i} \preceq {p_i}^{m'_i}$.
- $(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \preceq (\Sigma'; \Xi'_{\mathrm{L}} \blacktriangleleft \Xi'_{\mathrm{E}})$ *iff* $\Sigma \subseteq \Sigma'$, $\Xi_{\mathrm{L}} \preceq \Xi'_{\mathrm{L}}$ *and* $\Xi_{\mathrm{E}} \succeq \Xi'_{\mathrm{E}}$.

The conjunction *and* disjunction operators $\wedge^*$ *and* $\vee^*$ *denote the corresponding meet and join operators. When* $\Xi_1 \succeq \Xi_2$ *we say* Ξ_1 *is* weaker *than* Ξ_2*, and* Ξ_2 stronger *than* Ξ_1.

We put a mark in "$\wedge^*$" to distinguish it from "$\wedge$": "${p_1}^{m_1} \wedge {p_2}^{m_2}$" stands for the *multiplicity set* $\{{p_1}^{m_1}, {p_2}^{m_2}\}$ as described in the previous section, while "${p_1}^{m_1} \wedge^* {p_2}^{m_2}$" stands for the weakest ξ that satisfies $\xi \preceq {p_i}^{m_i}$ for both $i \in \{1, 2\}$. However the following lemma shows the distinction is actually unnecessary because as long as $p_1 \neq p_2$ those two processes are equal. We will therefore use "$\wedge$" indifferently for both senses of the symbol in the remainder of this paper.

Lemma 4.3.2 (Joining Disjoint Multiplicity Sets) *For all port pairs* $p \neq q$ *and multiplicities* m, m': $p^m \wedge^* q^{m'} = p^m \wedge q^{m'}$. *More generally, if* $\{p_i\}_{i\in I_1} \cap \{p_i\}_{i\in I_2} = \varnothing$ *then* $\left(\bigwedge_{i\in I_1} {p_i}^{m_i}\right) \wedge^* \left(\bigwedge_{i\in I_2} {p_i}^{m_i}\right) = \left(\bigwedge_{i\in I_1\cup I_2} {p_i}^{m_i}\right)$.

Proof Using Convention 4.2.1, $p^m \wedge^* q^{m'} \stackrel{\text{def}}{=} (p^m \wedge q^0) \wedge^* (p^0 \wedge q^{m'})$.

By definition of $\wedge^*$, $p^m \wedge^* p^0 = p^m$, so $(p^m \wedge q^0) \wedge^* (p^0 \wedge q^{m'}) = p^m \wedge q^{m'}$.

The generalisation is similar. □

We proceed with an operator that, given the types of two processes, computes the types of their parallel composition. This was done with $\wedge$ in the previous section when working with channel type mappings, but when working with multiplicities we need to *add* multiplicities.

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

Definition 4.3.3 (Multiplicity Addition and Subtraction) Multiplicity addition $m+m'$*, has* 0 *as a neutral element, and returns* $\star$ *for any pair of non-zero multiplicities.*

Multiplicity subtraction $m - m_1$ *is the largest* m_2 *such that* $m_1 + m_2 = m$*, using the ordering* $\forall m : 0 \leq m \leq \star$.

For instance, $1 + \omega = \star$ but $\star - 1 = \star$ and $\omega - 1$ is not defined because there is no m such that $m + 1 = \omega$. The following Lemma gives a direct way of computing subtraction:

Lemma 4.3.4 (Multiplicity Subtraction and Properties)

- $\forall m : \star - m = \star$
- $\forall m : m - 0 = m$.
- *If* $m \neq \star$, $m - m = 0$.
- *Unless covered in the above three cases,* $m - m'$ *is undefined.*
- *For all* m, m': $(m + m') - m'$ *is well-defined and larger or equal to* m.
- $(\{0, 1, \omega, \star\}; +)$ *is a monoid.*

We define composition on sets of multiplicities before lifting it to full process types.

Definition 4.3.5 (Multiplicity Set Composition, Subtraction) Composition *of multiplicity sets is done by the binary operator* $\odot$ *defined as follows, where* $p_i \neq p_j$ *for all* $i \neq j$*:*

$$\bigwedge_{i\in I} {p_i}^{m_i} \odot \bigwedge_{i\in I} {p_i}^{m'_i} \stackrel{\text{def}}{=} \bigwedge_{i\in I} {p_i}^{m_i+m'_i}$$

Subtraction *of multiplicity sets is done by the binary operator* $\setminus$ *and defined as follows, where* $p_i \neq p_j$ *for all* $i \neq j$*:*

$$\bigwedge_{i\in I} {p_i}^{m_i} \setminus \bigwedge_{i\in I} {p_i}^{m'_i} \stackrel{\text{def}}{=} \bigwedge_{i\in I} {p_i}^{m_i-m'_i}$$

The first point of Convention 4.2.1 permits generalising this definition for types with different domains of Σ. For instance:

$(a\wedge b)\odot(b\wedge c) = (a^1\wedge b^1\wedge c^0)\odot(a^0\wedge b^1\wedge c^1) = a^{1+0}\wedge b^{1+1}\wedge c^{0+1} = a^1\wedge b^\star\wedge c^1$, and similarly $(a^1\wedge b^\star\wedge c^1)\setminus(b^1\wedge c^1) = a^{1-0}\wedge b^{\star-1}\wedge c^{1-1} = a^1\wedge b^\star\wedge c^0 = a^1\wedge b^\star$.

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

Lemma 4.3.6 (Subtraction Properties) *For any three statements* Δ_1*,* Δ_2 *and* Δ_3*:*

$$\Delta_1 \setminus (\Delta_2 \odot \Delta_3) \cong (\Delta_1 \setminus \Delta_2) \setminus \Delta_3$$

The proof is given in Section A.1.6, as part of the proof of Lemma 4.3.8 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 4.3.7 (Process Type Composition — Multiplicities) *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*

$$\Gamma_1 \odot \Gamma_2 \stackrel{\text{def}}{=} (\Sigma_1 \wedge \Sigma_2 \,;\, \Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2} \blacktriangleleft (\Xi_{\mathrm{E}1} \setminus \Xi_{\mathrm{L}2}) \wedge^* (\Xi_{\mathrm{E}2} \setminus \Xi_{\mathrm{L}1}))$$

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

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

where $\Sigma = \{x:\lambda, y:\lambda, z:\lambda\}$, the local component is

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

and the environment component

$$\begin{aligned}(x^0\wedge\bar{x}\wedge y^0\wedge\bar{y})\setminus(\bar{y}\wedge\bar{z})\wedge(y\wedge\bar{y}^0\wedge z\wedge\bar{z}^0)\setminus(x\wedge y) =\\ (x^0\wedge\bar{x}\wedge y^0\wedge\bar{y}^{1-1})\wedge(y^{1-1}\wedge\bar{y}^0\wedge z\wedge\bar{z}^0).\end{aligned}$$

Lemma 4.3.8 (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.1.6. Lemma A.1.5 on page 65 gives more properties of the $\odot$ operator.

We now generalise the $\sigma[\tilde{x}]$ operator given in Definition 3.2.1 to apply on types with multiplicities, as a first step to similaly generalising the transition operator of Definition 3.2.3.

Special care is taken in case two x_i may be equal by separating all parameters, doing the substitution and then composing the resulting process types with the $\odot$ composition operator, much like (6).

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

Definition 4.3.9 (Channel Type Slicing) *The i^{th}* slice *of a channel type, written $\xi|_i$ where i is a parameter number, is a process type inductively defined as follows:*

$(i^m \wedge \bar{\imath}^{m'} \wedge \bigwedge_{j\in J} {p_j}^{m_j})|_i \stackrel{\text{def}}{=} i^m \wedge \bar{\imath}^{m'}$ *where* $\forall j \in J : \mathrm{n}(p_j) \neq i$.

$\langle \tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}} \rangle |_i \stackrel{\text{def}}{=} (i : \sigma_i; \xi_{\mathrm{I}}|_i \blacktriangleleft \xi_{\mathrm{O}}|_i)$.

Definition 4.3.10 (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 4.3.11 (Channel Instantiation) *Let σ be an n-ary channel type. Let $\tilde{x}$ be a sequence of n names.*

Input-instantiating *σ 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 *σ 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 $\sigma = \langle \lambda\lambda; \bar{1} \wedge 2; 1 \wedge \bar{2} \rangle$. Then

$$\sigma[x, y] = (x : \lambda; \bar{x} \blacktriangleleft x) \odot (y : \lambda; y \blacktriangleleft \bar{y}) = (x : \lambda, y : \lambda; \bar{x} \wedge y \blacktriangleleft x \wedge \bar{y})$$

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 σ, $\sigma[x, x] = (x : \lambda; \bar{x} \blacktriangleleft x) \odot (x : \lambda; x \blacktriangleleft \bar{x}) = \langle x : \lambda; x \wedge \bar{x}; x^0 \wedge \bar{x}^0 \rangle$. 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 $x^0, \bar{x}^0$ part. For example, $a(xy).(\bar{x} \,|\, y) \mid (\boldsymbol{\nu} b)\, \bar{a}\langle bb \rangle.\mathbf{0} \xrightarrow{\tau} \bar{b} \,|\, b$, where b's linearity is respected.

Proceeding to the transition operator, we start with a definition for transitions without parameters:

Definition 4.3.12 (p-Reduction — Multiplicities) *Let Γ be a process type and p a port. Then the $\Gamma \wr p$ operation is inductively defined as follows:*

- $p^m \wr q = \begin{cases} \text{undefined} & \text{if } p = q \text{ and } m = 0 \\ p^0 & \text{if } p = q \text{ and } m = 1 \\ p^m & \text{if } p \neq q \text{ or } m \in \{\omega, \star\} \end{cases}$

- $(\bigwedge_i \xi_i) \wr p \stackrel{\text{def}}{=} \bigwedge_i (\xi_i \wr p)$
- *If* $\Gamma = (\Sigma; \Xi_1 \blacktriangleleft \Xi_2)$ *then* $\Gamma \wr p \stackrel{\text{def}}{=} (\Sigma; \Xi_1 \wr p \blacktriangleleft \Xi_2 \wr \bar{p})$ *(if either of those two operations is undefined then so is* $\Gamma \wr p$*).*

On behavioural statements, $\Xi \wr p$ is similar but not quite the same as subtraction $\Xi \setminus p$[1] (Definition 4.3.7). The former simulates a transition, and in particular $p^\omega \wr p = p^\omega$ matches $!p \xrightarrow{p} !p$. 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 $!p \not\equiv p \,|\, !p$ as the right hand side has type $p^\star$).

As an illustration we show on (12) 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.

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 4.3.13 (Observability) *A port* p *is* observable *in a process type* Γ *(written* $\Gamma{\downarrow_p}$*) if* $\Gamma \wr p$ *is well-defined.*

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

Definition 4.3.14 (Output Composition) *The* output composition *operator* $\otimes$ *on process types is the binary operator such that* $\overline{\Gamma_1 \otimes \Gamma_2} = \overline{\Gamma_1} \odot \overline{\Gamma_2}$.

Finally, like in Definition 3.2.3 we need to ensure channel type annotations on bound output objects match the subject type. We write $\Gamma \wedge \Sigma$ to mean $\Gamma \odot (\Sigma; \top \blacktriangleleft \top)$.

We are now ready to generalise Definition 3.2.3 to process types:

Definition 4.3.15 (Transition Operator — Multiplicities) $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *being a process type with* $\Sigma(a) = \sigma$*, the effect of a transition* μ *on* Γ *is* $\Gamma \wr \mu$*, defined as follows.*

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

With the following Lemma we show that our goal of having channel types preserved in time and space is satisfied.

Lemma 4.3.16 (Channel Types are Constant) *Let Γ be a process type with channel type mapping Σ and $a \in \mathrm{dom}(\Sigma)$.*

For any Γ' such that $\Gamma_2 = \Gamma \odot \Gamma'$ is well defined, Σ_2 being Γ_2's channel type mapping, $\Sigma(a) = \Sigma_2(a)$.

For any μ such that $\Gamma_2 = \Gamma \wr \mu$ is well defined, Σ_2 being Γ_2's channel type mapping, $\Sigma(a) = \Sigma_2(a)$.

The Lemma is an immediate consequence of $(\Sigma \wedge \Sigma')$ being well-defined only if Σ and Σ' give equal types for common names (Definition 3.2.2).

4.4 Semantics

The semantics of process types with respect to a given process follows Definition 3.3.3 but with Γ instead of Σ, permits loosening of environment multiplicities, and enforces uniformity of ω:

Definition 4.4.1 (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 Γ_+ and μ' such that $(\Gamma_+; P') \xrightarrow{\mu'} (\Gamma_+ \wr \mu'; P'')$ for some P'', where μ' is obtained from μ 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 $p^{\omega} \succeq \Xi_{\mathrm{L}}$ 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 (4) by valid ones and some remote multiplicities are replaced by $\star$ to be able to inspect the components of τ-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, $(\left(l : \lambda; l^1 \wedge \bar{l}^0 \blacktriangleleft l^0 \wedge \bar{l}^0\right); P)$ is not correct because then $\Gamma_+ \wr \bar{l} = \bot$ and thus $P \xrightarrow{\bar{l}}$ has no corresponding transition from $(\Gamma_+; P)$.

Point 3 enforces uniform availability [San99] of ω 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 4.4.2 (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.2.1 for the proof.

Lemma 4.4.3 (Semantics Compositionality) *Let* $\Gamma_i \models_\# P_i$ *for both* $i = 1, 2$*, and* $\Gamma = \Gamma_1 \odot \Gamma_2$ *be well defined. Then* $\Gamma \models_\# P_1 \mid P_2$*.*

We conjecture that the corresponding property holds for all semantic definitions used in this paper, but the proofs becomes difficult for types involving dependency statements (Chapter 6). Soundness of the type systems, however, implies that property whenever both Γ_i are accepted by the type system for P_i.

4.5 Type System

The type system rules are nearly identical to the ones for arity given in Table 3, except that they work with process types rather than plain channel type mappings, and (as a base for computing multiplicities) the prefix rule contains an extra term for adding subject multiplicities.

Definition 4.5.1 (Multiplicity Type System) Typability *of a typed process* $(\Sigma; P)$ *with respect to* multiplicity*, written* $\Sigma \vdash_\# P$*, is inductively given by the rules in Table 4.*

Note how the arity type system was idempotent in its treatment of parallel composition and replication, that is P, $P \mid P$ and $!P$ were all treated the same way. This is of course no longer the case in the multiplicity type system which is all about counting name occurrences. The counting in the $P \mid P$ case is implemented through the $\odot$ operator. When typing a replicated process $!\,a(\tilde{y}).P$ or $!\,(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle.P$, $\#(G)$ is ω and parts of the type related to parameters or to the continuation must be *replicated*.

Lemma 4.5.2 (Existence of Replication) *Let* Γ *be a process type. Then there is a natural number* n *such that either* Γ^n *is not well defined or* $\Gamma^n \cong \Gamma^{n+1}$ *(where* $\Gamma^1 \stackrel{\text{def}}{=} \Gamma$ *and* $\Gamma^{n+1} \stackrel{\text{def}}{=} \Gamma^n \odot \Gamma$*).*

In the system exposed in this section, $n = 2$ always satisfies the requirements, but as we enrich the types in the next sections, larger numbers may become necessary.

This lemma implies that repeatedly composing a process type with itself reaches a eventually stabilises, which gives a practical way to compute "Γ^ω", a fix point of the $x \mapsto (x \odot \Gamma)$ function.

Definition 4.5.3 (Process Type Replication) *Replication of a process type* Γ *is defined with the following rule:* $\Gamma^\omega \stackrel{\text{def}}{=} \Gamma^n$ *where* n *is a value satisfying the lemma above.*

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

Much like the arity type system the (M-Res) rule *binds* the channel in the process type with a generalisation of Definition 3.4.2, which amounts to forcing it not to be used in the environment, i.e. setting its environment multiplicities to zero:

$$\frac{-}{(\varnothing; \top \blacktriangleleft \top) \vdash_{\#} \mathbf{0}} \text{ (M-Nil)} \qquad \frac{\Gamma \vdash_{\#} P \qquad \Gamma(x) = \sigma}{(\boldsymbol{\nu} x)\,\Gamma \vdash_{\#} (\boldsymbol{\nu} x : \sigma)\,P} \text{ (M-Res)}$$

$$\frac{\forall i : \Gamma_i \vdash_{\#} P_i}{\Gamma_1 \odot \Gamma_2 \vdash_{\#} P_1 \mid P_2} \text{ (M-Par)} \qquad \frac{\forall i : (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash_{\#} S_i}{\left(\bigwedge_i \Sigma_i; \bigvee_i^* \Xi_{\mathrm{L}i} \blacktriangleleft \bigwedge_i^* \Xi_{\mathrm{E}i}\right) \vdash_{\#} S_1 + S_2} \text{ (M-Sum)}$$

$$\frac{\Gamma \vdash_{\#} P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x}}{\begin{array}{rl} \left(p : \sigma;\ \blacktriangleleft p^m \wedge \bar{p}^{m'}\right) & \odot \\ \left(; p^{\#(G)} \blacktriangleleft\right) & \odot \\ (\boldsymbol{\nu}\mathrm{bn}(G))\left(\Gamma\right. & \odot \\ \left.\overline{\sigma}[\tilde{x}]\right)^{\#(G)} & \vdash_{\#} G.P \end{array}} \text{ (M-Pre)}$$

Table 4: Multiplicity Type System Rules

Definition 4.5.4 (Binding — Multiplicities) *The binding operator* $(\boldsymbol{\nu} x)$ *acts on sets of multiplicities as follows:*

$$(\boldsymbol{\nu} x) \bigwedge_{i \in I} {p_i}^{m_i} \stackrel{\text{def}}{=} \bigwedge_{i \in I : \mathrm{n}(p_i) \neq x} {p_i}^{m_i}$$

Binding a name x *in a process type* Γ *is perfomed as follows:*

$$(\boldsymbol{\nu} x)\ (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \stackrel{\text{def}}{=} ((\boldsymbol{\nu} x)\,\Sigma; (\boldsymbol{\nu} x)\,\Xi_{\mathrm{L}} \blacktriangleleft (\boldsymbol{\nu} x)\,\Xi_{\mathrm{E}})$$

Properties. The properties given on page 10 are generalised as follows:

Proposition 4.5.5 (Subject Congruence) *Let* $\Gamma \vdash_{\#} P \equiv P'$. *Then* $\Gamma \vdash_{\#} P'$.

This is a consequence of Proposition 7.3.1 on page 51.

Proposition 4.5.6 (Subject Reduction) *Let* $(\Gamma; P)$ *be a typed process such that* $\Gamma \vdash_{\#} P$. *Then, for any transition* $(\Gamma; P) \xrightarrow{\mu} (\Gamma \wr \mu; P')$, $\exists \Gamma'$ *s.t.* $\Gamma' \preceq \Gamma \wr \mu$ *and* $\Gamma' \vdash_{\#} P'$.

This is a consequence of Proposition 7.3.2, proved in Section A.3.

Proposition 4.5.7 (Decidability) *There is a decidable algorithm that, given a channel type mapping* Σ *and a process* P, *either constructs a process type* $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *where* $\Gamma \vdash_{\#} P$ *or, if there is no such* Γ, *rejects the process.*

This is a consequence of Proposition 7.3.3, proved on page 51.

Proposition 4.5.8 (Type Soundness) *If* $\Gamma \vdash_{\#} P$ *then* $\Gamma \models_{\#} P$.

This is a consequence of Proposition 7.3.4, proved in Section A.5.

4.6 Type System Tuning

Examining the type system, one can notice that one rule is not completely specified, namely (M-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 multiplicities.

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

- The simplest way is to always set $m = m' = \star$ in (M-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 $\left(\left(a^1 \wedge \bar{a}^1 \wedge b^1 \wedge b^0 \blacktriangleleft a^\star \wedge \bar{a}^\star \wedge b^\star \wedge \bar{b}^\star\right) \vdash P\right)$
- The other extreme is to run the type system twice, first to record the multiplicities obtained in Ξ_{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 $\left(\left(a^1 \wedge \bar{a}^1 \wedge b^1 \wedge \bar{b}^0 \blacktriangleleft a^0 \wedge \bar{a}^0 \wedge b^0 \wedge \bar{b}^0\right) \vdash P\right)$
- 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 variants is chosen it may at times be desirable to override the default behaviour, for instance through annotations in the process.

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 of 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, process (1), after sending the $\overline{a}\langle t' f' \rangle$ query, performs a *branching* between t' and f' guided by the reply to its request, using the "+" π-calculus operator. If a replies through f', the process *selects* $\bar{f}$ out of the set of admissible behaviours $\bar{t}$ and $\bar{f}$. If a replies through t', (1) *delegates* the selection to b.

We now extend the types defined in the previous section to describe choices made by processes. The key idea is to introduce a $\vee$-connective into multiplicity sets. Due to the similarity with a logical statement we call them behavioural statements:

Definition 5.0.1 (Behavioural statement — Choice) *A* behavioural statement *is an expression of the form*

$$\xi, \Xi \;::=\; p^m \;\;\big|\;\; (\Xi \vee \Xi) \;\;\big|\;\; (\Xi \wedge \Xi) \;\;\big|\;\; \bot \;\;\big|\;\; \top \tag{13}$$

where ξ ranges over statements containing only parameter numbers *and Ξ over statements containing channel* names*.*

Intuitively, *selection* $\Xi_1 \vee \Xi_2$ is correct if one of the Ξ_i is, *conjunction* $\Xi_1 \wedge \Xi_2$ if both Ξ_i are. $\top$ always holds and $\bot$ never does. The formal semantics follow Definition 4.4.1, but where the operators have been extended to deal with disjunction, as covered in the following section.

5.1 Algebra

The weakening relation given in Definition 4.3.1 is generalised to permit increasing dependencies as well as rearranging items in a behavioural statement, much like $\equiv$ works on processes.

Definition 5.1.1 (Weakening Relation — Choice) *Relation $\preceq$ is the smallest preorder defined by the following rules, where $\cong$ is its symmetric subset. When $\Xi_1 \cong \Xi_2$, we say Ξ_1 are* $\cong$-equivalent *or just* equivalent*.*

1. *On multiplicities, for all p and m, $p^0 \preceq p^m \preceq p^\star \cong \top$.*
2. *On multiplicity sets:*
   - *$\Xi_1 \wedge \Xi_2 \preceq \Xi_1 \preceq \Xi_1 \vee \Xi_2$, and $\bot \preceq \Xi \preceq \top$.*
   - *$\Xi \wedge (\Xi_1 \vee \Xi_2) \cong (\Xi \wedge \Xi_1) \vee (\Xi \wedge \Xi_2)$ and $\Xi \vee (\Xi_1 \wedge \Xi_2) \cong (\Xi \vee \Xi_1) \wedge (\Xi \vee \Xi_2)$*
   - *$\wedge$ and $\vee$ are commutative, associative and idempotent, up to $\cong$.*
   - *If $\Xi_1 \preceq \Xi_2$ then $\Xi \wedge \Xi_1 \preceq \Xi \wedge \Xi_2$ and $\Xi \vee \Xi_1 \preceq \Xi \vee \Xi_2$.*
   - *The $\cong$ relation is a congruence, and $\succeq$ is covariant with respect to $\vee$ and $\wedge$.*

From the above rules one derives some useful properties:

Lemma 5.1.2 (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 ε 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 following Lemma formally states that process types may be considered up to $\cong$ (see also Lemma 4.4.2):

Lemma 5.1.3 (Types may be seen up to $\cong$) *Let Ξ_1, Ξ_2 be behavioural statements such that $\Xi_1 \cong \Xi_2$.*

Let $\Phi[\cdot]$ be some expression involving behavioural statements, using only the $\odot$, $\otimes$, $\backslash$, $\wr$ operators and with one "hole" $[\cdot]$. Then $\Phi[\Xi_1] \cong \Phi[\Xi_2]$.

Now let $\Phi[\cdot]$ be some statement involving behavioural statements and the above operators, as well as any of the relations $\cong$ and $\preceq$. Then $\Phi[\Xi_1]$ is true iff $\Phi[\Xi_2]$ is.

The proofs are given in Section A.1.1.

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

Definition 5.1.4 (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 $\Gamma_i = (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i})$,*

$$\Gamma_1 \ \vee \ \Gamma_2 \stackrel{\text{def}}{=} (\Sigma_1 \wedge \Sigma_2; \Xi_{\mathrm{L}1} \vee \Xi_{\mathrm{L}2} \blacktriangleleft \Xi_{\mathrm{E}1} \wedge \Xi_{\mathrm{E}2})$$

$$\Gamma_1 \ \wedge \ \Gamma_2 \stackrel{\text{def}}{=} (\Sigma_1 \wedge \Sigma_2; \Xi_{\mathrm{L}1} \wedge \Xi_{\mathrm{L}2} \blacktriangleleft \Xi_{\mathrm{E}1} \vee \Xi_{\mathrm{E}2})\,.$$

The $\wedge$ operator on mappings Σ_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. We defer the definition of this notion and the proof that every type has an equivalent normal form (Lemma 6.3.3) until introducing behavioural statements including dependencies.

We now modify the transition operator to work with disjunction. p-reduction $\Gamma \wr p$ is a logical homomorphism such that $p^0 \wr p \stackrel{\text{def}}{=} \bot$ and otherwise follows definition 4.3.12. As $p^0 \wr p$ produces the neutral element $\bot$ of selection ($\varepsilon \vee \bot \cong \varepsilon$) rather than failing, impossible elements in a selection are pruned when information about the process state gets revealed by transition labels. For instance assume the type Γ 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).

Definition 4.3.9 applies only to behavioural statements without disjunctions, we show in the following definition how to slice arbitrary statements.

Definition 5.1.5 (Channel Type Slicing — Choice) Slicing *a channel type behavioural statement not using selection, 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, $\langle \tilde{\sigma}; \xi_I; \xi_O \rangle\,|_i \stackrel{\text{def}}{=} (i : \sigma_i; \xi_I|_i \blacktriangleleft \xi_O|_i)$.

Defining $\wedge$ and $\vee$ for channel types as in Definition 5.1.4,

$(\sigma_1 \vee \sigma_2)|_i \stackrel{\text{def}}{=} \sigma_1|_i \vee \sigma_2|_i$ *and* $(\sigma_1 \wedge \sigma_2)|_i \stackrel{\text{def}}{=} \sigma_1|_i \wedge \sigma_2|_i$ *gives the general case.*

Subtraction of behavioural statements (Definition 4.3.5) is generalised as follows:

Definition 5.1.6 (Process Subtraction) *The Subtraction Operator $\setminus$ is given by:*

- $\bigwedge_{i\in I} {p_i}^{m_i} \setminus \bigwedge_{i\in I} {p_i}^{m'_i} \stackrel{\text{def}}{=} \bigwedge_{i\in I} {p_i}^{m_i - m'_i}$
- *for a set of Ξ_i and Ξ'_j not using disjunction:*

$$\bigvee_{i\in I} \Xi_i \setminus \bigvee_{j\in J} \Xi'_j \stackrel{\text{def}}{=} \bigvee_{\rho: J\to I} \bigwedge_{j\in J} (\Xi_{\rho(j)} \setminus \Xi'_j)$$

- *when no other rules apply, $\Xi \setminus \Xi' = \Xi$.*

In the second point, ρ ranges over all functions with domain J — they do not need to be surjective or injective.

Existence of replication (Lemma 4.5.2 that permits defining the replication operator Γ^ω) still holds with processes including arbitrary behavioural statements, so Definition 4.5.3 applies with no changes to such process types. We omit the proof but it can be shown in two parts: If Γ doesn't use disjunction then $n = 2$ satisfies the requirements (although $n = 1$ may also work for some processes) and Γ^2 replaces all non-zero multiplicities by $\star$. Secondly, if Γ's normal form[4] 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.

5.2 Semantics

Generalising of semantics (Definition 4.4.1) to deal with process types with choice has been taken care of by generalising the type algebra, specifically the $\wr$ operator.

5.3 Type System

The only change to the type system in this iteration is a new (Sum) rule, where separate branches of a sum are reflected as separate terms of a disjunction, so for instance $a + b$ is typed with a type having $(a^1 \wedge b^0) \vee (a^0 \wedge b^1)$ as local component.

$$\frac{\forall i : (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash_\# G_i.P_i}{(\bigwedge_i \Sigma_i; \bigvee_i \Xi_{\mathrm{L}i} \blacktriangleleft \bigwedge_i \Xi_{\mathrm{E}i}) \vdash_\# \sum_i G_i.P_i} \text{ (C-Sum)}$$

Other rules are as in Table 4 on page 22.

The exact properties given on page 22 hold for this type system as well, except that Subject Congruence only holds up to $\cong$:

Proposition 5.3.1 (Subject Congruence) *Let $\Gamma \vdash_\# P$ and $P \equiv P'$. Then $\Gamma' \vdash_\# P'$ for some $\Gamma \cong \Gamma'$.*

[4] General definition in Lemma 6.3.3.

6 Activeness and Responsiveness

6.1 Introduction

A common requirement one may wish to express about a component written in π-calculus is that a process should be listening (respectively, ready to send) at an input (resp., output) port. Let us call this property *activeness* at a port.[5] We first review our needs before proceeding to a formal definition.

For example, assuming the process language contains pair constructs and a deconstructor, consider a process P decoding a value v and sending a signal on a channel s, and a process Q first sending a signal and then decoding v.

$$\begin{aligned} P &= a(v).\mathsf{case}\, v\, \mathsf{of}\, (x,y) : \bar{s} \\ Q &= a(v).\bar{s}.\mathsf{case}\, v\, \mathsf{of}\, (x,y) : \mathbf{0} \end{aligned}$$

These processes could be encoded in our language as follows, where u holds an encoding of v.

$$\begin{aligned} [\![P]\!] &= a(u).\overline{u}(\boldsymbol{\nu} r_1 r_2).r_1(x).r_2(y).\bar{s} \\ [\![Q]\!] &= a(u).\bar{s}.\overline{u}(\boldsymbol{\nu} r_1 r_2).r_1(x).r_2(y).\mathbf{0} \end{aligned}$$

It is simple to show that $P \sim Q$; however, $[\![P]\!] \not\approx [\![Q]\!]$, since these processes are distinguished by

$$R \;=\; \overline{a}(\boldsymbol{\nu} u).\bot.!\, u(xy).(\overline{x}\langle b\rangle | \overline{y}\langle c\rangle) \tag{14}$$

where

$$\bot.P \stackrel{\text{def}}{=} (\boldsymbol{\nu} t)\, t.P \tag{15}$$

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

Before we propose a solution, it should be noted that requiring u to be active is not enough for P and Q to be indistinguishable, as is shown by

$$R \;=\; \overline{a}(\boldsymbol{\nu} u).!\, u(xy).\bot.(\overline{x}\langle b\rangle | \overline{y}\langle c\rangle) \tag{16}$$

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 specifying in u's channel type that its parameter must itself be active and requiring u to be *responsive* in addition to active, i.e. provide the resources declared in the channel type with no additional dependencies.

Moreover, in order to have a property which is meaningful for nonlinear names we add a *reliability* requirement to activeness. Consider the process $P = p(x).\bar{x}$, where p is plain (i.e. has multiplicities $p^\star \wedge \bar{p}^\star$). 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 like $E = \overline{p}\langle c\rangle.\mathbf{0}$ "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).

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

Finally, our target being encodings, there will be typically an overhead (in terms of extra τ-transitions) in an encoded process compared to the original one. Therefore it is acceptable if a number of τ-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 and responsiveness:

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

1. *P will eventually (i.e. possibly after a finite number of τ-reductions) contain an unguarded occurrence of p in subject position.*
2. *The port is "reliable": no third party can prevent p from being made available to a process attempting to communicate with that port.*

A process P is said input responsive *on a channel a with type σ given ε if any input on a yields a process providing all resources in σ's input component, depending at most on ε and resources in σ's output component. The process is said* output *responsive if the above holds when "input" and "output" are swapped.*

6.2 Dependency Statements

In this section we extend behavioural statements (Definition 5.0.1 on page 23) with behavioural properties denoted $\mathbf{A}$ and $\mathbf{R}$ (ranged over by k), *resources* p_k (ranged over by α, β, γ) and *dependency statements* "$\gamma \triangleleft \varepsilon$". We first motivate the new notions and then present the new grammar.

A resource $p_{\mathbf{A}}$ means 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 σ_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

$$\sigma_r = \langle \sigma_p; 1^\star \wedge \bar{1}^\star; 1^\star \wedge \bar{1}^\star \rangle$$

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 $\sigma_u = \langle \sigma_r, \sigma_r; \bar{1}_{\mathbf{A}} \wedge \bar{2}_{\mathbf{A}}; 1_{\mathbf{A}} \wedge 2_{\mathbf{A}} \rangle$, 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 $\langle \sigma_u; \bar{1}^\star; 1^\omega_{\mathbf{A}} \wedge \bar{1}^\star \rangle$, where both input and output ports of a may send requests on the parameter u but a's *output* port must provide one replicated ("ω") 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.

Resources can be *conditional* on other resources, which is expressed with *dependency statements*. Intuitively, $\Xi \triangleleft \Xi'$ holds in a process P if whenever Ξ' holds in E, Ξ holds in $P \mid E$.

What is the difference between "$\Xi_1 \blacktriangleleft \Xi_2$" and "$\Xi_1 \triangleleft \Xi_2$"? The former says two things: the process behaves like Ξ_1, and the environment is required to satisfy Ξ_2. The second statement says that if the environment satisfies Ξ_2 then the process will satisfy Ξ_1. For instance assume some process P satisfies one of those two statements ($\Xi_1 \blacktriangleleft \Xi_2$ or $\Xi_1 \triangleleft \Xi_2$). Then composing P with a process Q gives a process $P \,|\, Q$ satisfying Ξ_1 if Q satisfies Ξ_2. If Q does *not* satisfy Ξ_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.

Definition 5.0.1 on Page 23 is thus generalised to include dependency statements:

Definition 6.2.1 (Behavioural statement)

$$\xi, \Xi \;::=\; p^m \;\mid\; (\Xi \vee \Xi) \;\mid\; (\Xi \wedge \Xi) \;\mid\; \bot \;\mid\; \top \;\mid\; \gamma \triangleleft \varepsilon \tag{17}$$

$$\varepsilon \;::=\; \gamma \;\mid\; (\varepsilon \vee \varepsilon) \;\mid\; (\varepsilon \wedge \varepsilon) \;\mid\; \bot \;\mid\; \top \tag{18}$$

$$\alpha, \beta, \gamma \;::=\; s_{\mathbf{A}} \;\mid\; p_{\mathbf{R}} \tag{19}$$

We extend Convention 4.2.1 with items specific to dependency statements:

Convention 6.2.2 (Notation for Behavioural Statements)

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* $\triangleleft$ *is right-distributive:*

   - $(\Xi_1 \vee \Xi_2) \triangleleft \Xi \stackrel{\text{def}}{=} (\Xi_1 \triangleleft \Xi) \vee (\Xi_2 \triangleleft \Xi)$,
   - $(\Xi_1 \wedge \Xi_2) \triangleleft \Xi \stackrel{\text{def}}{=} (\Xi_1 \triangleleft \Xi) \wedge (\Xi_2 \triangleleft \Xi)$,
   - $\top \triangleleft \Xi \stackrel{\text{def}}{=} \top$.
   - *Multiplicities* p^m *may not have dependencies, so* $p^m \triangleleft \varepsilon \stackrel{\text{def}}{=} p^m$.

3. $p_{\mathbf{AR}}$ *abbreviates* $(p_{\mathbf{A}} \wedge p_{\mathbf{R}})$*, and* $p^m_{\mathbf{A}} \triangleleft \varepsilon$ *means* $(p^m \wedge p_{\mathbf{A}} \triangleleft \varepsilon)$*.*

4. *A dependency "*$\triangleleft \top$*" can be omitted.*

5. *In channel types, and in the local component of process types, any port whose responsiveness dependencies are not specified are understood to be responsive without dependencies.*

6. *In addition, the local component* Ξ_{L} *of a process type with channel type mapping* Σ *should be understood as follows:*

$$\Xi_{\mathrm{L}} \wedge \bigwedge_{x \in \mathcal{N} \setminus \mathrm{dom}(\Sigma)} \left(x^0 \wedge \bar{x}^0 \wedge x_{\mathbf{R}} \wedge \bar{x}_{\mathbf{R}} \right)$$

Like for $p^\star$, the goal of statements such as $p_{\mathbf{R}} \triangleleft \bot$ (literally, "if *falsity* holds, then p is responsive"), logically equivalent to $\top$, is just to prevent point 5 in the above convention from applying.

Examples. We present below some process type examples.

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

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

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

2. The type $\big(a : \lambda; a^{\star}_{\mathbf{A}} \blacktriangleleft a^0 \wedge \bar{a}^{\star}\big)$ is a valid description of process $\mathbf{!}\,a.\mathbf{0}$, but not of $a.\mathbf{0}$, because the latter is unreliable. The type $\big(a : \lambda; a^{\star}_{\mathbf{A}} \blacktriangleleft a^0 \wedge \bar{a}^1\big)$, 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.

3. Finally, using the notation

$$?.P \stackrel{\text{def}}{=} (\boldsymbol{\nu} t)\,(\bar{t} \,|\, t \,|\, t.P) \tag{20}$$

(assuming t fresh) as a shortcut for an "unreliable prefix", the type $\big(a : \langle \lambda; \bar{1}_{\mathbf{A}}; 1_{\mathbf{A}}\rangle; a_{\mathbf{A}} \blacktriangleleft a^0 \wedge \bar{a}\big)$ is a valid description of process $a(x).\bar{x}$, 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 $\big(a : \langle \lambda; \bar{1}_{\mathbf{A}}; 1_{\mathbf{A}}\rangle; a \blacktriangleleft a^0 \wedge \bar{a}\big)$ 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.

4. The input port of a Boolean channel (such as r, a, and b in process (1), Page 2) has type

$$\bar{1}^1_{\mathbf{A}} \;\vee\; \bar{2}^1_{\mathbf{A}}, \tag{21}$$

that says that either the first parameter ("1") must be output ("$\bar{1}$") active ("$_{\mathbf{A}}$"), and the second parameter unused,[6] or ("$\vee$") the opposite.

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

$$\bar{b}(\boldsymbol{\nu} tf).(t.P + f.Q) \tag{22}$$

is a responsive client (correctly implementing "if b then P else Q"). An alternative implementation using internal choice may lead to deadlocks. Let the internal choice operator $\oplus$ be

$$P \oplus Q \stackrel{\text{def}}{=} (\boldsymbol{\nu} u)\,\big(\bar{u} \,|\, (u.P + u.Q)\big) \tag{23}$$

for some $u \notin (\mathrm{fn}(P) \cup \mathrm{fn}(Q))$. Process

$$\bar{b}(\boldsymbol{\nu} tf).(t.P \oplus f.Q) \tag{24}$$

may evolve to the "wrong" branch, eg. to $t.P$ whereas the environment respond on f.

[6] Remember Convention 4.2.1 on page 15: ports that aren't mentioned have multiplicity zero.

We want (22) to be recognised as correct and (24) 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 of resources. In Definition 6.2.1, the third rule (19) is substituted by the following one.

$$\alpha, \gamma \ ::= \ p_{\mathbf{R}} \ \mid \ s_{\mathbf{A}} \tag{25}$$

$$s \ ::= \ p \ \mid \ (p + s) \tag{26}$$

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:

$$\left(1^1 \ \vee \ 2^1\right) \wedge (1 + 2)_{\mathbf{A}}, \tag{27}$$

which is similar to (21) 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 (22) and broken by (24).

The Boolean type gathers (21) and (27) as

$$\mathsf{Bool} \stackrel{\text{def}}{=} \left\langle \lambda, \lambda \, ; \ \bar{1}^1_{\mathbf{A}} \ \vee \ \bar{2}^1_{\mathbf{A}} \ ; \ (1^1 \vee 2^1) \wedge (1 + 2)_{\mathbf{A}} \right\rangle \tag{28}$$

Consider the following process:

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

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

By definition, $a_{\mathbf{R}} \triangleleft \bar{x}_{\mathbf{A}}$ (a is responsive if $\bar{x}$ is active), which gives us $a_{\mathbf{R}} \triangleleft \bar{u}_{\mathbf{A}}$. 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:

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

Now we have $a_{\mathbf{A}} \triangleleft (\bar{t}_{1\mathbf{A}} \vee \bar{t}_{2\mathbf{A}})$ and $a_{\mathbf{R}} \triangleleft (\bar{u}_{1\mathbf{A}} \wedge \bar{u}_{2\mathbf{A}})$: 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:

$$\overline{a}\langle t \rangle.!\, b(x).\bar{x} \mid !\, a(y).\bar{b}\langle y \rangle \tag{29}$$

We have both $b_{\mathbf{A}} \triangleleft a_{\mathbf{A}}$ (because of the left-hand component) and $a_{\mathbf{R}} \triangleleft b_{\mathbf{R}}$ (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 "$a \triangleleft b \triangleleft a$" and have the process rejected.

We can now add activeness annotation to (12) as a type for the process (1). The local behavioural statement states that r is active with multiplicity ω (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.

$$\Gamma_A = \big(a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}; \\ \big(r^{\omega} \wedge (r_{\mathbf{A}} \triangleleft \top)\big) \wedge \big(r_{\mathbf{R}} \triangleleft (a_{\mathbf{A}} \wedge a_{\mathbf{R}} \wedge b_{\mathbf{A}} \wedge b_{\mathbf{R}})\big) \blacktriangleleft a^{\omega} \wedge b^{\omega} \wedge r^{0}\big) \quad (30)$$

By convention 4.2.1, the previous type can be rewritten:
$\big(a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}; r^{\omega}_{\mathbf{A}} \wedge r_{\mathbf{R}} \triangleleft (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \blacktriangleleft a^{\omega} \wedge b^{\omega} \wedge r^{0}\big)$

We introduce some syntactic restrictions on what channel types are acceptable:

Definition 6.2.3 (Channel Type Restrictions) *A channel type $\langle \tilde{\sigma}; \xi_I; \xi_O \rangle$ is said to have* shared activeness *(between its input and its output) if there is an activeness resources $p_{\mathbf{A}}$ such that both $p^{m}_{\mathbf{A}} \triangleleft \varepsilon \in \xi_I$ and $p^{m'}_{\mathbf{A}} \triangleleft \varepsilon' \in \xi_O$ for some m, m', ε and ε'.*

The type is said to have blocked activeness *if there is a port p such that either $p^{m}_{\mathbf{A}} \triangleleft \varepsilon \in \xi_I$ and $\bar{p}^{0} \in \xi_O$, or $p_{\mathbf{A}} \triangleleft \varepsilon \in \xi_O$ and $\bar{p}^{0} \in \xi_I$.*

We also say it is a case of blocked activeness if there is a term $p^{0}_{\mathbf{A}}$ in either ξ_I or ξ_O.

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

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

$$\sigma = \langle \lambda, \lambda; \bar{1}^{\star}_{\mathbf{A}} \wedge 1^{\star} \wedge 2_{\mathbf{A}} \triangleleft \bar{1}_{\mathbf{A}}; \bar{1}^{\star}_{\mathbf{A}} \wedge 1^{\star} \wedge \bar{2}_{\mathbf{A}} \triangleleft \bar{1}_{\mathbf{A}} \rangle$$

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

$$a(xy).x.y \quad (31)$$

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 $\overline{a}\langle bc \rangle.(!\,\bar{b} \,|\, b.\bar{c})$. 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 (31) being accepted: indeed, the protocol requires the output to provide an x-output without conditions, and the input in (31) 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 activeness simultaneously require one side of the channel to provide activeness on a port, and forbid the other side to connect to that port.

The reason for ruling out such types is that analysing processes such as $\overline{a}\langle a\rangle$ with $a : \sigma = \langle \sigma; \top; \bar{1}^{\star}_{\mathbf{A}} \wedge 1^{\star}\rangle$ 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 activeness avoids such paradoxical cases.

Finally, a valid receiver (or sender) on a channel type with unstable multiplicities may become invalid through a τ-transition, by consuming its own parameters. For instance, having $a : \sigma = \langle \lambda; \ldots; 1 \wedge \bar{1}\rangle$, $P = (\boldsymbol{\nu} b)\,(\overline{a}\langle b\rangle \,|\, b \,|\, \bar{b})$ 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 paper, the following:

Convention 6.2.4 *No channel types involved in a semantic judgement or in a typing judgement may have shared or blocked activeness, or unstable multiplicities (this also applies to parameter types at all depths).*

6.3 Algebra

We start by refining the notions of weakening of (process) types and of normal forms.

Definition 6.3.1 (Weakening Relation)

1. *On dependencies and behavioural statements (together ranged over by η):*
   - $\eta_1 \wedge \eta_2 \preceq \eta_1 \preceq \eta_1 \vee \eta_2$, *and* $\bot \preceq \eta \preceq \top$ $\eta \wedge (\eta_1 \vee \eta_2) \cong (\eta \wedge \eta_1) \vee (\eta \wedge \eta_2)$.
   - $\wedge$ *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* $\eta \vee \eta_1 \preceq \eta \vee \eta_2$
   - *If* $\eta_1 \cong \eta_2$ *then* $\gamma \triangleleft \eta_1 \cong \gamma \triangleleft \eta_2$, $(\eta \blacktriangleleft \eta_1) \cong (\eta \blacktriangleleft \eta_2)$ *and* $(\eta_1 \blacktriangleleft \eta) \cong (\eta_2 \blacktriangleleft \eta)$
2. *On dependency statements:*
   - $(\gamma \triangleleft \varepsilon_1) \wedge (\gamma \triangleleft \varepsilon_2) \cong \gamma \triangleleft (\varepsilon_1 \vee \varepsilon_2)$
   - $(\gamma \triangleleft \varepsilon_1) \vee (\gamma \triangleleft \varepsilon_2) \cong \gamma \triangleleft (\varepsilon_1 \wedge \varepsilon_2)$
   - $\gamma \triangleleft \bot \cong \top$

Deciding if two types are equivalent or related by weakening can be done by constructing their *normal forms*.

Lemma 6.3.2 (Normal Forms) *Any behavioural statement or dependency ε can be written in a* normal form *ε' with the following properties:*

1. $\varepsilon' \cong \varepsilon$.
2. $\varepsilon' = \bigvee_{i \in I} \varepsilon_i$ *and* $\varepsilon_i = \bigwedge_{j \in I_j} u_{ij}$ *where u_{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.*

3. *The sets I and I_j are minimal.*

All ε also have "conjunctive normal forms" defined the same way but with the disjunction and conjunction swapped in point 2.

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 $\gamma \triangleleft \varepsilon_1$ and $\gamma \triangleleft \varepsilon_2$ match as f.i. $(\gamma \triangleleft \varepsilon_1) \vee (\gamma \triangleleft \varepsilon_2) \cong \gamma \triangleleft (\varepsilon_1 \wedge \varepsilon_2)$.

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

In the former case, $\bigwedge_{i \in I} \Xi_i \vee \bigwedge_{i' \in I'} \Xi_{i'} \cong \bigwedge_{i' \in I'} \Xi_{i'}$ (as a consequence of $(\Xi_1 \cong \Xi_2) \Rightarrow \big((\Xi_1 \vee \Xi_2) \cong \Xi_1\big)$). In the latter case, $\bigwedge_{i \in I} \Xi_i \vee \bigwedge_{i' \in I'} \Xi_{i'} \cong \bigwedge_{i \in I \setminus \{\hat{\imath}\}} \Xi_i \wedge (\Xi_{\hat{\imath}} \vee \Xi_{\hat{\imath}'})$ (as a consequence of the $(\Xi \wedge \Xi_1) \vee (\Xi \wedge \Xi_2) \cong \Xi \wedge (\Xi_1 \vee \Xi_2)$ rule).

Note that matching is symmetric and reflexive but *not* transitive. For instance $\alpha \triangleleft \varepsilon_\alpha \wedge \beta \triangleleft \varepsilon_\beta \wedge \gamma \triangleleft \varepsilon_\gamma$ matches both $\alpha \triangleleft \varepsilon_\alpha \wedge \beta \triangleleft \hat{\varepsilon}_\beta \wedge \gamma \triangleleft \varepsilon_\gamma$ and $\alpha \triangleleft \varepsilon_\alpha \wedge \beta \triangleleft \varepsilon_\beta \wedge \gamma \triangleleft \hat{\varepsilon}_\gamma$ 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 Ξ has such a normal form.

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

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

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

The above proof applies to conjunctive normal forms if $\top$ and $\bot$ are swapped, all disjunctions occurring in the proof are replaced by conjunctions and the other way round. □

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

Lemma 6.3.3 (Weakening Decidability) *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 $i \in I$, there is $j \in J$ such that $\varepsilon_i \preceq \varepsilon_j$.*
2. $\bigwedge_{i \in I} \varepsilon_i \preceq \bigwedge_{j \in J} \varepsilon_j$ *if for all $j \in J$, there is $i \in I$ such that $\varepsilon_i \preceq \varepsilon_j$.*

See Section A.1.3 for the proof.

Behavioural statement composition $\Xi \odot \Xi'$ builds on two auxiliary operators: A *dependency reduction* operator that collapses dependency chains, and a *cleaning* operator that drops unused portions of process types.

The reduction relation, like weakening, highlights the logical aspect of behavioural statements and is analogous to the *modus ponens* rule in logic. Parallel composition may create dependency chains which must then be reduced. For example $a.\bar{b}$ and $b.\bar{c}$ satisfy respectively $\bar{b}_{\mathbf{A}} \triangleleft \bar{a}_{\mathbf{A}}$ and $\bar{c}_{\mathbf{A}} \triangleleft \bar{b}_{\mathbf{A}}$, while their composition $a.\bar{b} \,|\, b.\bar{c}$, in addition to $\bar{b}_{\mathbf{A}} \triangleleft \bar{a}_{\mathbf{A}}$, satisfies $\bar{c}_{\mathbf{A}} \triangleleft (\bar{a}_{\mathbf{A}} \vee \bar{b}_{\mathbf{A}})$.

More generally:

Definition 6.3.4 (Dependency Reduction) *The* reduction relation $\hookrightarrow$ *on behavioural statements is a partial order relation satisfying the following rules where channel type mappings Σ have been omitted for clarity.*

1. $(s_{\mathbf{A}} \triangleleft \varepsilon) \wedge (\gamma \triangleleft \varepsilon') \;\hookrightarrow\; (s_{\mathbf{A}} \triangleleft \varepsilon) \wedge (\gamma \triangleleft \varepsilon' \{^{\varepsilon\{^{\perp}/_{\gamma}\} \vee s_{\mathbf{A}}}/_{s_{\mathbf{A}}}\})$
2. $(p_{\mathbf{R}} \triangleleft \varepsilon) \wedge (\gamma \triangleleft \varepsilon') \;\hookrightarrow\; (p_{\mathbf{R}} \triangleleft \varepsilon) \wedge (\gamma \triangleleft \varepsilon' \{^{\varepsilon\{^{\perp}/_{\gamma}\} \wedge p_{\mathbf{R}}}/_{p_{\mathbf{R}}}\})$
3. *for $m \neq 0$, $p \neq q$ and $\varepsilon \not\cong \perp \not\cong \varepsilon'$:* $(p + q + s)_{\mathbf{A}} \triangleleft \varepsilon \;\wedge\; p_{\mathbf{A}} \triangleleft \varepsilon' \;\wedge\; \bar{q}^m \hookrightarrow \perp$

   On process types:
4. $\Xi \hookrightarrow \Xi'$ *implies* $(\Xi \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow (\Xi' \blacktriangleleft \Xi_{\mathrm{E}})$ *and* $(\Xi_{\mathrm{L}} \blacktriangleleft \Xi) \hookrightarrow (\Xi_{\mathrm{L}} \blacktriangleleft \Xi')$.
5. $(p_{\mathbf{R}} \triangleleft \varepsilon_1 \blacktriangleleft p_{\mathbf{R}} \triangleleft \varepsilon_2) \hookrightarrow (p_{\mathbf{R}} \triangleleft (\varepsilon_1 \wedge \varepsilon_2) \blacktriangleleft p_{\mathbf{R}} \triangleleft \varepsilon_2)$
6. *If* $(\alpha \triangleleft \varepsilon) \preceq \Xi_{\mathrm{E}}$ *with* $\beta \preceq \varepsilon$ *then* $(\gamma \triangleleft \varepsilon' \blacktriangleleft \Xi_{\mathrm{E}}) \hookrightarrow$ $\big(\gamma \triangleleft (\varepsilon' \{^{\alpha \wedge \beta}/_{\alpha}\}) \blacktriangleleft \Xi_{\mathrm{E}}\big)$ *for* $\beta \neq \gamma$.
7. *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 $C[\cdot]$.[7]

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

Point 3 simulates a selection and a branching occurring inside a process, by replacing every term of the branching that does not match the selection by $\perp$, which is the neutral element of $\vee$. For example the transition

$$\bar{t} \mid (t.P + f.Q) \xrightarrow{\tau} P$$

is matched by

$$\begin{aligned}
(t+f)_{\mathbf{A}} \wedge \big((t_{\mathbf{A}} \wedge \Gamma_P) \vee (f_{\mathbf{A}} \wedge \Gamma_Q)\big) \wedge \bar{t}^1 \quad &\cong \quad \big((t+f)_{\mathbf{A}} \wedge t_{\mathbf{A}} \wedge \Gamma_P \wedge \bar{t}^1\big) \vee \\
&\qquad \big((t+f)_{\mathbf{A}} \wedge f_{\mathbf{A}} \wedge \Gamma_Q \wedge \bar{t}^1\big) \\
&\hookrightarrow \quad \big((t+f)_{\mathbf{A}} \wedge t_{\mathbf{A}} \wedge \Gamma_P \wedge \bar{t}^1\big) \vee \perp \\
&\cong \quad \big((t+f)_{\mathbf{A}} \wedge t_{\mathbf{A}} \wedge \Gamma_P \wedge \bar{t}^1\big)
\end{aligned}$$

We require activeness of the branching to prevent the rule from applying in case there is a risk of race conditions.

Point 4 and 7 permit applying reduction on any part of a process type.

Points 5 and 6 permit reduction 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

[7] I.e. $C \;::=\; [\cdot] \;\mid\; C \wedge \Xi \;\mid\; C \vee \Xi$

expected remote behaviour from the type. For instance $\overline{a}\langle x\rangle.x.\bar{s}$, where a is alternating, may have reduced $\bar{s}_{\mathbf{A}} \triangleleft \bar{x}_{\mathbf{A}}$ and $\bar{x}_{\mathbf{A}} \triangleleft a_{\mathbf{R}}$ into $\bar{s}_{\mathbf{A}} \triangleleft a_{\mathbf{R}}$. Simulating the $\overline{a}\langle x\rangle$ transition effectively cancels the $\bar{x}_{\mathbf{A}} \triangleleft a_{\mathbf{R}}$ term and the reduction it caused, as the environment component of $\overline{\sigma}[x] \triangleleft (\bar{a}_{\mathbf{R}} \blacktriangleleft a_{\mathbf{R}})$ contains $x_{\mathbf{A}} \triangleleft a_{\mathbf{R}}$ which, through rule 6, replaces $a_{\mathbf{R}}$-dependencies by $\bar{x}_{\mathbf{A}}$ dependencies. Similarly, the $\bar{x}_{\mathbf{A}} \triangleleft a_{\mathbf{R}}$ statement becomes $\bar{x}_{\mathbf{A}} \triangleleft \bar{x}_{\mathbf{A}}$, i.e. $\bar{x}_{\mathbf{A}} \triangleleft \bot$.

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

Lemma 6.3.5 (Closure Uniqueness) *Every behavioural statement has, up to $\cong$ (Definition 6.3.1), exactly one closure.*

The proof is given in Section A.1.4 on page 66 and includes an algorithm for computing the closure.

The different treatment of activeness and responsiveness (in γ's dependencies, the former gets a $\vee$ and the latter a $\wedge$), can be understood as follows: If two processes P_1 and P_2 both provide an a-input, it is enough that one of them is able to receive a request to have a active in $P_1|P_2$. On the other hand, they must both be responsive in order to guarantee that all a-requests will get a response. Also note how *self-dependencies* $\gamma \triangleleft \gamma$ are replaced by $\gamma \triangleleft \bot$. Activeness self-dependencies are found in deadlocks such as $\bar{a}.!\, b \,|\, \bar{b}.!\, a$ where $a_{\mathbf{A}}$ and $b_{\mathbf{A}}$ depend on each other, and responsiveness self-dependencies are found in livelocks such as $!\, a(x).\bar{b}\langle x\rangle \,|\, !\, b(x).\overline{a}\langle x\rangle$ where $a_{\mathbf{R}}$ and $b_{\mathbf{R}}$ depend on each other.

The second auxiliary operator for process type composition is the following, that drops parts of process types that are no longer used after a composition:

Definition 6.3.6 (Removal of Non-Observable Dependencies) *Let Γ 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 Ξ_{L} as many times as possible. In the following, s ranges over $\left(\sum_{i\in I} p_i\right)$ where $i \neq j$ both in I implies $p_i \neq p_j$. Whenever $k = \mathbf{R}$, I contains a single index.*

- *Replace any statement $s_k \triangleleft \varepsilon$ where at most one p_i is observable (Definition 4.3.13) in Γ by $\top$*
- *In any statement $\gamma \triangleleft \varepsilon$, replace any $s_{\mathbf{A}}$ in ε by $\bot$ whenever at least one p_i is not observable in Γ's complement $\overline{\Gamma}$, and any $s_{\mathbf{R}}$ by $\top$.*

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 $\Gamma = \left(\Sigma; t \vee f \blacktriangleleft \bar{t} \vee \bar{f}\right)$, 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*.

The p-reduction operator (page 18) is extended as follows to deal with dependency statements.

Definition 6.3.7 (p-Reduction) *The p-reduction operator on processes containing dependency statements is the logical homomorphism defined by the following rule in addition to those given in Definition 4.3.12:*

$\left(\sum_i p_i\right)_{\mathbf{A}} \wr p \stackrel{\text{def}}{=} \top$ *if both $p = p_i$ and $p \neq p_{i'}$ for some $i \neq i'$.*

We first generalise composition of behavioural statements, before lifting it to full process types.

Definition 6.3.8 (Behavioural Statement Composition)
The behavioural statement composition *operator* $\odot$ *is the logical homomorphism such that:*

1. $(p^m) \odot (p^{m'}) \stackrel{\text{def}}{=} p^{m+m'}$
2. $(s_{\mathbf{A}} \triangleleft \varepsilon) \odot (s_{\mathbf{A}} \triangleleft \varepsilon') \stackrel{\text{def}}{=} (s_{\mathbf{A}} \triangleleft \varepsilon) \wedge (s_{\mathbf{A}} \triangleleft \varepsilon')$
3. $(p_{\mathbf{R}} \triangleleft \varepsilon) \odot (p_{\mathbf{R}} \triangleleft \varepsilon') \stackrel{\text{def}}{=} (p_{\mathbf{R}} \triangleleft \varepsilon) \vee (p_{\mathbf{R}} \triangleleft \varepsilon')$
4. $\Xi \odot \bot \stackrel{\text{def}}{=} \bot$
5. *When no other rule applies,* $\Xi \odot \Xi' \stackrel{\text{def}}{=} \top$.

Note again the difference in behaviour between activeness and responsiveness, very similar to the one occurring in behavioural statement reduction. Indeed it is a simple exercise to verify that dependency reduction commutes with composition or, more accurately:

Lemma 6.3.9 *For any two behavioural statements* Ξ *and* Ξ'*:*

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

Point 5 above and Convention 6.2.2 interact in a subtle way to give the following property:

Lemma 6.3.10 (Composition of disjoint statements) *For two statements* Ξ *and* Ξ'*, having no resources in common when written according to Convention 6.2.2 (specifically, its point 5),* $\Xi \odot \Xi' = \Xi \wedge \Xi'$

The proof is given in Section A.1.5 on page 68.

Definition 6.3.11 (Process Type Composition) *Process Type* composition $\Gamma_i = (\Sigma_i; \Xi_{\text{L}i} \blacktriangleleft \Xi_{\text{E}i})$ *with* $i = 1, 2$*, written* $\Gamma_1 \odot \Gamma_2$

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

When performing channel type instantiation, slicing of channel types including dependency statements follows Definition 4.3.9 with the following additional rule: $\gamma \triangleleft \varepsilon|_i = \begin{cases} \gamma \triangleleft \varepsilon & \text{if } \mathrm{n}(\gamma) = i \\ \top & \text{otherwise} \end{cases}$ Note that the i^{th} term may now include, in the dependencies, resources of parameters other than i.

Definition 6.3.12 (Transition Operator) $\Gamma = (\Sigma; \Xi_{\text{L}} \blacktriangleleft \Xi_{\text{E}})$ *being a process type with* $\Sigma(a) = \sigma$*, the effect of a transition* μ *on* Γ *is* $\Gamma \wr \mu$*, defined as follows.*

- $\Gamma \wr \tau \stackrel{\text{def}}{=} \Gamma$,
- $\Gamma \wr a(\tilde{x}) \stackrel{\text{def}}{=} \Gamma \wr a \odot \sigma[\tilde{x}] \triangleleft (a_{\mathbf{R}} \blacktriangleleft \bar{a}_{\mathbf{R}})$,

- $\Gamma \wr (\boldsymbol{\nu} \tilde{z} : \tilde{\sigma})\, \overline{a}\langle \tilde{x} \rangle \stackrel{\text{def}}{=} \Gamma \wr \bar{a} \otimes \bar{\sigma}[\tilde{x}] \triangleleft (\bar{a}_{\mathbf{R}} \blacktriangleleft a_{\mathbf{R}})$.

In the above definition, $\Gamma \triangleleft (\bar{a}_{\mathbf{R}} \blacktriangleleft a_{\mathbf{R}})$ makes Γ's local component depend on $\bar{a}_{\mathbf{R}}$ and its environment component depend on $a_{\mathbf{R}}$. We illustrate the above operator on the process A (Equation 1):

We illustrate the transition operator on the process A (Equation 1):

The transition $A \xrightarrow{r(uv)} A' = A \,|\, \overline{a}(\boldsymbol{\nu} t' f').(t'.\bar{b}\langle uv \rangle + f'.\bar{v})$ is matched on Γ_A (30) by

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

The "$\wr r$" part has no effect, as discussed when illustrating Definition 4.3.12. Computing the composition works as follows, where the numbers match those in Definition 6.3.11.

1. The channel type mapping of the resulting process type is just

   $$a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}, u : \lambda, v : \lambda$$

   The local component is

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

   and the environment component is just the conjunction.

   $$\big(a^{\omega} \wedge b^{\omega} \wedge r^{0}\big) \;\wedge\; \big((u + v)_{\mathbf{A}} \triangleleft \bar{r}_{\mathbf{R}} \wedge (u \vee v)\big)$$

2. Closure of the resulting expression reduces the dependency chain

   $$(\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \triangleleft r_{\mathbf{R}} \;\wedge\; r_{\mathbf{R}} \triangleleft (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}})$$

   producing the statement

   $$(\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \triangleleft (r_{\mathbf{R}} \wedge a_{\mathbf{AR}} \wedge b_{\mathbf{AR}})$$

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

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

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

$$\begin{aligned}\big(a : \mathsf{Bool}, b : \mathsf{Bool}, r : \mathsf{Bool}, u : \lambda, v : \lambda; \\ (\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \triangleleft (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \blacktriangleleft \\ a^{\omega} \wedge b^{\omega} \wedge r^{0} \wedge (u \vee v)\big)\end{aligned} \quad (32)$$

as a type for $A \,|\, \overline{a}(\boldsymbol{\nu} t' f').(t'.\bar{b}\langle uv \rangle + f'.\bar{v})$, 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 Γ_A.

6.4 Semantics

We now outline semantic definitions for the liveness properties "**A**" and "**R**", using the notation $\Gamma \models P$, read "The process type Γ is a correct description of process P".

We first define *immediate* correctness of a statement, that tells whether it holds for the process in its current state.

Definition 6.4.1 (Immediate Correctness) *A dependency statement $\gamma \lhd \varepsilon$ is* immediately correct *in a typed process $(\Gamma; P)$ if it satisfies one of the following rules.*

1. *A dependency statement $\gamma \lhd \bot$ is always immediately correct.*
2. *An activeness statement $(\sum_{i\in I} p_i)_{\mathbf{A}} \lhd \varepsilon$ is immediately correct if*
   $P \equiv (\boldsymbol{\nu}\tilde{z})\left((\sum_{j\in J} G_j.C_j) \mid Q\right)$ *with* $I \subseteq J$, *and*
   $\forall i \in I\,(\mathsf{sub}(G_i) = p_i$ *and* $\mathrm{n}(p_i) \notin \tilde{z})$.
3. *A responsiveness statement $p_{\mathbf{R}} \lhd \varepsilon$ is immediately correct if, for any transition $(\Gamma; P) \xrightarrow{(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle}$ s.t. $\mathrm{n}(p) \in \tilde{x} \setminus \tilde{z}$, it is the case that $\overline{\sigma}[\tilde{x}]|_{p_{\mathbf{R}}} = p_{\mathbf{R}} \lhd \varepsilon_0$ with $(a_{\mathbf{R}} \wedge \varepsilon_0) \succeq \varepsilon$.*

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 we do not allow more than one transition to test it. However when an output process *delegates* 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 point about responsiveness in the above definition checks.

6.4.1 Type Projections

In addition to the reasons just presented, we also need a way to simulate selections done by the environment. Consider the statement

$$(\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \lhd (\bar{t}'_{\mathbf{A}} \vee \bar{f}'_{\mathbf{A}}). \tag{33}$$

which applies to $A|B$ where A is (1) and B is a process active and responsive at b, after a request has been sent to r and a has been queried: if the environment provides one of $\bar{t}'_{\mathbf{A}}$ or $\bar{f}'_{\mathbf{A}}$, then the process will provide one of $\bar{t}_{\mathbf{A}}$ or $\bar{f}_{\mathbf{A}}$. In other words the remote side is permitted to select one of t' and f', and for any of them, the local side is permitted to select one of t and f. As the way the process provides $(\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}})$ differs depending which of $\bar{t}'_{\mathbf{A}}$ or $\bar{f}'_{\mathbf{A}}$ is made available, we use a *projection relation* that performs the selection, and may for instance project (33) to $(\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \lhd \bar{f}'_{\mathbf{A}}$, which can then be shown true through the $\xrightarrow{f'} \xrightarrow{\bar{f}}$ transition sequence.

Any behavioural statement Ξ_2 can be written as a conjunction of disjunctions (Lemma 6.3.2, and then any of those disjunctions (Ξ_0 in the definition below) is called a projection of Ξ_2. Formally:

Definition 6.4.2 (Elementary Statement and Projections)
An elementary statement *is a behavioural statement of the form*

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

Let Ξ_2 be a behavioural statement. An elementary statement Ξ_0 is a projection *of Ξ_2, written $\Xi_2 \searrow \Xi_0$, iff $\Xi_2 \succeq \Xi_0$ and, for all elementary statements Ξ_1 such that $\Xi_2 \succeq \Xi_1 \succeq \Xi_0$, we have $\Xi_0 \cong \Xi_1$.*

Remember (Definition 6.3.1, point 2 on page 33) 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 described by

$$(\gamma_1 \triangleleft (\alpha_{11} \vee \alpha_{12})) \wedge (\gamma_2 \triangleleft (\alpha_{21} \vee \alpha_{22}))$$

It has four projections, one of which is $\gamma_2 \triangleleft \alpha_{21}$, which corresponds to the environment requesting γ_2, and providing α_{21} in exchange.

6.4.2 Correctness Strategies

The informal definition of activeness (Definition 6.1.1) says that a process will *eventually* be ready to communicate at a port. A difficulty in defining "eventually" is that it depends on the scheduler. We chose a definition that assumes some fairness from the scheduler (to be able to state results in presence of divergence) but not too much (so that for instance the results hold with a stochastic scheduler).

A natural semantic definition of a conditional statement $\gamma \triangleleft \varepsilon$ for a typed process $(\Gamma_1; P)$ would be "for all correctly typed processes $(\Gamma_2; P_2)$ such that ε is included in Γ_2 and $\Gamma_1 \odot \Gamma_2$ is well defined, $(\Gamma_1 \odot \Gamma_2; P_1 \mid P_2)$ satisfies γ."

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 $\bigvee_i \left(\gamma_i \triangleleft \bigwedge_j \alpha_{ij}\right)$ 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 γ_i is immediately available. We call such a path a *strategy*.

Definition 6.4.3 (Strategy Function)

1. *A* strategy function *f maps typed processes to pairs of transition labels and typed processes such that*

   if $f(\Gamma; P) = (\mu; \Gamma'; P')$ then $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$.

2. *A* strategy relation $\xrightarrow{f}$ *on typed processes is defined by the following rules:*

   (a) $f(\Gamma; P) = (\mu; \Gamma'; P')$ *implies* $(\Gamma; P) \xrightarrow{f} (\Gamma'; P')$, *and*

   (b) $(\Gamma; P) \xrightarrow{f} (\Gamma; P)$ *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.

6.4.3 Satisfaction

The actual semantic definition is stated similarly to the usual notion of *fairness*, which says, for a scheduler, that if a particular transition is constantly available, it will eventually occur . Instead of a particular transition we use a strategy function. The definition is structured in three parts. The first numbered list defines some symbols that build up a "test" of the strategy. The second numbered list specifies what constitutes a valid test and the last numbered list provides conditions for the test to be passed.

Definition 6.4.4 (Satisfaction) *Let* Γ *be a type and* P *a process. We say that* P satisfies Γ *(or* Γ *is* correct *for* P*), written* $\Gamma \models P$*, if* $\Gamma \models_{\#} P$ *and there is a strategy* f *s.t. for any*

1.1 *"query" transition sequence* $\tilde{\mu}_q$ *and* $(\Gamma_0; P_0)$ *with* $(\Gamma; P) \xrightarrow{\tilde{\mu}_q} (\Gamma_0; P_0)$,

1.2 *sequence of typed processes* $\{(\Gamma_i; P_i)\}_{i \in \mathbb{N}_0}$,

1.3 *infinite "strategy application point" set* $I \subseteq \mathbb{N}_0$,

1.4 *finite indexing set* J *and*

1.5 *collection of resources* $\{\gamma_j\}_{j \in J}$ *and dependencies* $\{\varepsilon_{i,j}\}_{i \in \mathbb{N}_0, j \in J}$

such that

2.1 $\forall i \in I$*:* $(\Gamma_i; P_i) \searrow (\Gamma'_i; P_i) \xrightarrow{f} (\Gamma_{i+1}; P_{i+1})$ *for some* Γ'_i*, and*

2.2 $\forall i \in (\mathbb{N}_0 \setminus I) : (\Gamma_i; P_i) \searrow (\Gamma'_i; P_i) \xrightarrow{\mu_i} (\Gamma_{i+1}; P_{i+1})$ *for some* Γ'_i, μ_i.

2.3 *For all* $i \in \mathbb{N}_0$ *let* $\Xi'_{\mathrm{L}i}$ *be* Γ'_i*'s local dependency network. Then* $\Xi'_{\mathrm{L}0} \cong \bigvee_{j \in J} \gamma_j \triangleleft \varepsilon_{j,0}$ *and* $\forall i > 0$*:* $\Xi'_{\mathrm{L}i} \preceq \bigvee_{j \in J} \gamma_j \triangleleft \varepsilon_{j,i}$,

we have $j \in J$ *such that:*

3.1 *For each* $i \in I$ *such that* $(\Gamma_i; P_i) \xrightarrow{f} (\Gamma_{i+1}; P_{i+1})$ *is a transition with subject* p_i, $\varepsilon_{j,i} \preceq \overline{p_i}_{\mathbf{A}}$

3.2 *If* γ_j *is an activeness resource there is a number* $i \in I$ *such that* $\gamma_j \triangleleft \varepsilon_{j,i}$ *is immediately correct for* $(\Gamma_i; P_i)$.

3.3 *If* γ_j *is a responsiveness resource then for all* $i \in \mathbb{N}_0$, $\gamma_j \triangleleft \varepsilon_{j,i}$ *is immediately correct for* $(\Gamma_i; P_i)$.

Although the $\{P_i\}_{i\in\mathbb{N}_0}$ sequence must be infinite, it may correspond to a finite number of (strong) transitions if, after some point, all μ_i are empty and the strategy does no transition.

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 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 "we have j 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 i s.t.".

The semantics of responsiveness is mostly provided by the transition operator when computing Γ_0 from Γ and $\tilde{\mu}_q$. For example, to verify a statement $a_{\mathbf{R}} \triangleleft \varepsilon$, one can submit the process to a sequence of transitions ending in an input $a(\tilde{x})$, after which the transition operator will introduce $\sigma[\tilde{x}] \triangleleft \varepsilon$ into the type (σ being a's type), so that the behaviour specified in the channel type must then be provided by the process in order for satisfaction to hold.

The following formalises the intuition behind weakening:

Lemma 6.4.5 (Bisimulations and Type Equivalence) *Let* $(\Gamma; P)$ *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 Section A.2.2 for the proof. We need the check for correct multiplicities because uniformity is not always preserved by bisimilarity. On the other hand we have the following corollary derived from Lemma 4.4.2 which lets us drop the condition on $\Gamma' \models_{\#} P'$, and also justifies our identifying types up to $\cong$ and processes up to $\equiv$. See also Proposition 7.3.1.

Corollary 6.4.6 *Let* $\Gamma \preceq \Gamma'$ *with* $\Gamma \models P$, *and* $P \equiv P'$. *Then* $\Gamma' \models P'$ *as well.*

6.4.4 Example

We now sketch a proof that Γ_A given in (30), page 32, is a correct type for $P = A$ given in (1).

We prove a few values of $(\Gamma'_0; P_0)$; others are similar. For the sake of readability we omit channel type mappings and environment multiplicities. It is easy to see that, the r-input being replicated and having multiplicity ω, $A \xrightarrow{\tilde{\mu}_q} P_0$ implies $P_0 \equiv A \,|\, R$ for some R, and similarly $\Gamma_0 = \Gamma_A \wr \tilde{\mu}_q$ must be $\cong$-equivalent to $\Gamma_A \wedge \Gamma_R$ for some Γ_R disjoint from Γ_A in the sense of Lemma 6.3.10 and so any $\Gamma_0 \searrow \Gamma'_0$ (required by 2.1 and 2.2) satisfies either $\Gamma_A \searrow \Gamma'_0$ or $\Gamma_R \searrow \Gamma'_0$. We focus on the former first.

1. Let $\Gamma_0 \searrow \Gamma'_0 = r_{\mathbf{A}}$. No matter which sequence is selected, as Γ'_0 contains no disjunction, J has a single element $\hat{\jmath}$ and $\forall i \in \mathbb{N}_0 : \varepsilon_{\hat{\jmath},i} = \top$. Define the strategy f to be *idle* in this sequence (i.e. $\forall (r_{\mathbf{A}}; P_i) : (r_{\mathbf{A}}; P_i) \xrightarrow{f} (r_{\mathbf{A}}; P_i)$).

   Point 3.1 is vacuously true as f does no labelled transitions. We have $\forall i : \Gamma_i = r_{\mathbf{A}}$ and $P_i \equiv A \,|\, R_i$ for some R_i. The index i needed in point 3.2 of the Definition can be set to any value from I because $r_{\mathbf{A}}$ is always immediately correct in a process $A \,|\, R_i$.

2. Let instead $\Gamma_0 \searrow \Gamma'_0 = r_{\mathbf{R}} \triangleleft (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}})$. Here again $J = \{\hat{\jmath}\}$ and $\gamma_{\hat{\jmath}} = r_{\mathbf{R}}$. This case is again shown correct by having f be idle and $r_{\mathbf{R}} \triangleleft \varepsilon_{\hat{\jmath},i}$ is also always immediately correct (Definition 6.4.1) by virtue of r never appearing in object position in any $P_i \xrightarrow{\mu_i} P_{i+1}$ if $\Gamma'_i \wr \mu_i$ is to be well-defined.

We now study two representative transition sequences projecting on Γ_R, and discuss the strategy f as we go forward.

3. We start by sending a request $\tilde{\mu}_q = r(uv)$ to the process. Its projection Γ'_0 is given by (32) on page 38 which is elementary ($\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{R}}$ contains no "$\wedge$" and $a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}$ contains no "$\vee$"). J contains two elements, let's call them 1 and 2, such that $\gamma_1 = \bar{u}_{\mathbf{A}}$ and $\gamma_2 = \bar{v}_{\mathbf{A}}$, so the strategy's work is to bring activeness on one of those two ports.

4. To bring the process closer to that goal we set the strategy to follow the $P_0 \xrightarrow{\bar{a}(\boldsymbol{\nu} t' f')} A \,|\, (t'.\bar{b}\langle uv\rangle + f'.\bar{v})$ output, which is permitted (point 3.1 of the Definition) because its subject $\bar{a}$ has its complement a active in both the dependencies of $\bar{u}_{\mathbf{A}}$ and $\bar{v}_{\mathbf{A}}$ (although is is not yet decided which one will be available). The local behavioural statement is now

$$\Gamma_1 = (\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \triangleleft (a_{\mathbf{AR}} \wedge (\bar{t}'_{\mathbf{A}} \vee \bar{f}'_{\mathbf{A}}) \wedge b_{\mathbf{AR}})$$

5. If we do not want to help the strategy find the way out we set $1 \in I$ as well. As specified by point 2.1 we must still pick a projection $\Gamma_1 \searrow \Gamma'_1$ 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 $\bar{f}'$:

$$\Gamma'_1 = (\bar{u}_{\mathbf{A}} \vee \bar{v}_{\mathbf{A}}) \triangleleft (a_{\mathbf{AR}} \wedge \bar{f}'_{\mathbf{A}} \wedge b_{\mathbf{AR}})$$

6. The process is now

$$P_1 = A \,|\, (t'.\bar{b}\langle uv\rangle + f'.\bar{v})$$

   so the strategy must consume the f' prefix, which is permitted because its complement is active ($\bar{f}'_{\mathbf{A}}$) in the dependencies, while a t'-transition would violate the condition fixed in point 3.1 of the Definition.

7. We are now at the process $P_2 = A|\bar{v}$. If $2 \in I$ then $i = 2$ satisfies the requirement 3.2 as $\bar{v}_{\mathbf{A}}$ is now immediately correct. If instead we consume $\bar{v}$ with $\mu_2 = \bar{v}$, the transition operator removes activeness of both $\bar{u}$ and $\bar{v}$ (see Definition 6.3.4 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 6.4.1, is always immediately correct for any $i > 2$.

$$\frac{-}{(\varnothing; \top \blacktriangleleft \top) \vdash \mathbf{0}} \text{ (R-Nil)}$$

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

$$\frac{\begin{array}{c}\forall i : \big(\mathsf{sub}(G_i) = \{p_i\}, \quad (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash G_i.P_i\big) \\ \Xi_{\mathrm{E}} \preceq \bigwedge_i \Xi_{\mathrm{E}i} \\ \big(\Xi_{\mathrm{E}} \text{ has concurrent environment } p_{i'}\big) \Rightarrow \varepsilon = \bot\end{array}}{\big(\bigwedge_i \Sigma_i; (\sum_i p_i)_{\mathbf{A}} \triangleleft \varepsilon \wedge \bigvee_i \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}}\big) \vdash \sum_i G_i.P_i} \text{ (R-Sum)}$$

$$\frac{\begin{array}{c}\Gamma \vdash P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x} \\ (\#(G) = 1 \text{ and } m' = \star) \Rightarrow \varepsilon = \bot\end{array}}{\begin{array}{rl}\Big(p : \sigma;\ \blacktriangleleft p^m \wedge \bar{p}^{m'}\Big) & \odot \\ \Big(; p_{\mathbf{A}}^{\#(G)} \triangleleft \varepsilon \blacktriangleleft\Big) & \odot \\ (\boldsymbol{\nu}\mathrm{bn}(G))\Big(\ \Gamma \triangleleft \bar{p}_{\mathbf{A}} & \odot \\ \overline{\sigma}[\tilde{x}] \triangleleft \bar{p}_{\mathbf{AR}} & \odot \\ (; p_{\mathbf{R}} \triangleleft \sigma[\tilde{x}] \blacktriangleleft\ )\Big)^{\#(G)} & \vdash G.P\end{array}} \text{ (R-Pre)}$$

Table 5: Type System Rules

Picking $\bar{t}'$ instead of $\bar{f}'$ at step 5 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.

6.5 Type System

A typed process $(\Gamma; P)$ is *typable* according to the Activeness and Responsiveness Type System (written $\Gamma \vdash P$) if the judgement can be derived from the Activeness and Responsiveness Type System.

Definition 6.5.1 (Activeness and Responsiveness Type System) *The rules in Table 5 inductively define the Activeness and Responsiveness Type System.*

The next section presents an example and a detailed discussion of each rule.

6.6 A Typing Example

We illustrate the five factors of the (R-Pre) rule in order with the derivation of

$$r_{\mathbf{R}} \triangleleft (a_{\mathbf{AR}} \wedge b_{\mathbf{AR}}) \tag{34}$$

(r is responsive if both a and b are active and responsive) as a type for the process (1) on page 2. All rules of the type system except (R-Par) are used in this derivation so we will describe them in the order in which they are used. For an explanation of (R-Par), refer to the discussion of Definition 6.3.11.

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 (R-Nil) (that types the idle process with the neutral element of $\odot$).

Subject type, multiplicities and activeness. The parameter-less output $\bar{f}$ is typed using the prefix rule (R-Pre). The name is linear ($m = m' = 1$) and, since there are no parameters or continuation, all but the first two factors of the typing are empty, leaving us with: $\left(f : \lambda;\ \blacktriangleleft\ \bar{f}^1 \wedge f^1\right) \odot \left(;\, \bar{f}^1_{\mathbf{A}} \triangleleft \top \blacktriangleleft\ \right)$, that is:

$$\Gamma_{35} = \left(f : \lambda; \bar{f}_{\mathbf{A}} \blacktriangleleft\ \bar{f}^0 \wedge f^1\right) \vdash\ \bar{f} \tag{35}$$

Continuation. Sequence $G.P$ is typed much like parallel composition $G|P$, except that activeness resources in P additionally depend on 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 π-calculus produces the exact same type. For our process, $f'.\bar{f}$ is again typed with (R-Pre), where the first three terms are now non-null:

$$\left(f' : \lambda;\ \blacktriangleleft\ {f'}^1 \wedge \bar{f}'^1\right) \odot \left(;\, {f'}^1_{\mathbf{A}} \triangleleft \top \blacktriangleleft\ \right) \odot \Gamma_{35} \triangleleft \bar{f}'_{\mathbf{A}} \vdash\ f'.\bar{f}$$

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

$$\Gamma_T = \left(f : \lambda, f' : \lambda; \bar{f}_{\mathbf{A}} \triangleleft \bar{f}'_{\mathbf{A}} \blacktriangleleft\ \bar{f}^0 \wedge f^1 \wedge {f'}^0 \wedge \bar{f}'^1\right) \vdash\ f'.\bar{f} \tag{36}$$

Remote Behaviour. The fourth statement in the prefix rule plays two roles, respectively through the local and environment components of the instantiated channel $\overline{\sigma}[tf]$. First, it states that, if the input on a channel 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 (27) from page 31 with t and f replacing 1 and 2 (and omitting terms with a zero exponent). Second, as was already done by the type systems of the previous sections, it sets upper bounds on how many times the local side is permitted to use the parameters' ports. In this case we get $t^1 \vee f^1$ in the environment side, 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:

$$\left(b : \mathsf{Bool}, t : \lambda, f : \lambda; (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \triangleleft b_{\mathbf{AR}} \blacktriangleleft\ (t^1 \vee f^1) \wedge (\bar{b}^\star \wedge b^\omega)\right) \vdash\ \bar{b}\langle tf\rangle \tag{37}$$

As in (36), the t'-prefix adds a dependency on $\bar{t}'_{\mathbf{A}}$ to all activeness resources, effectively turning the $b_{\mathbf{AR}}$ dependency into $b_{\mathbf{AR}} \wedge \bar{t}'_{\mathbf{A}}$:

$$\Gamma_F = \left(\Sigma; (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \triangleleft (b_{\mathbf{AR}} \wedge \bar{t}'_{\mathbf{A}}) \blacktriangleleft\ (t^1 \vee f^1) \wedge (\bar{b}^\star \wedge b^\omega)\right) \vdash\ t'.\bar{b}\langle tf\rangle \tag{38}$$

Typing Sums. In the rule (R-Sum), a process type having no "concurrent environment $p_{i'}$" prevents a third-party process to attempt selecting more than one branch of the sum, and, by contraposition, guarantees that any attempt to select a branch of the sum (by communicating with its guard) will succeed, which is what activeness of the branching means.

Definition 6.6.1 (Concurrent Port Use) *Let $\{p_i\}_{i\in I}$ be a set of ports. Then:*

- *A behavioural statement Ξ has* concurrent p_i *if there is no $i \in I$ 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 Ξ or Ξ' has.*
- *A process type $(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ has concurrent environment p_i if and only if Ξ_{E} has concurrent p_i.*

A sum $T + F$ is given, through (R-Sum), the type $(t' + f')_{\mathbf{A}} \triangleleft \varepsilon \wedge (\Gamma_T \vee \Gamma_F)$, where Γ_T and Γ_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 Γ_T and Γ_F. The decoupling between the guards and the continuations is done to make explicit which channels must be used to make the process branch. In the example (1), in addition to $(t' + f')_{\mathbf{A}}$, the type for the continuation of the a-output is obtained from (36) and (38):

$$\begin{aligned}\big(\Sigma; (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \triangleleft (b_{\mathbf{AR}} \wedge \bar{t}'_{\mathbf{A}}) \vee (\bar{f}_{\mathbf{A}} \triangleleft \bar{f}'_{\mathbf{A}}) \blacktriangleleft \\ (t^1 \vee f^1) \wedge \bar{b}^\star \wedge b^\omega \wedge \bar{f}^0 \wedge {f'}^0 \wedge \bar{f}'^1\big) \vdash t'.\bar{b}\langle tf\rangle + f'.\bar{f}\end{aligned} \quad (39)$$

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

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

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

The $\odot$ operator now has to do dependency reduction (Definition 6.3.4 on page 35): The remote behaviour provides either $\bar{t}'_{\mathbf{A}} \triangleleft a_{\mathbf{AR}}$ or $\bar{f}'_{\mathbf{A}} \triangleleft a_{\mathbf{AR}}$, and in the continuation either $(\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}})$ depends on $t'_{\mathbf{A}}$, or $\bar{f}_{\mathbf{A}}$ depends on $f'_{\mathbf{A}}$. Remember, for activeness resources, if α depends on β and β on γ, then α depends on $(\beta \vee \gamma)$, so the two dependency statements in the continuation become respectively $(\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \triangleleft (t'_{\mathbf{A}} \vee a_{\mathbf{AR}})$ and $\bar{f}_{\mathbf{A}} \triangleleft (f'_{\mathbf{A}} \vee a_{\mathbf{AR}})$.

Binding. The binding operator (Definition 4.5.4) acts on dependency statements as follows:

Definition 6.6.2 (Binding) *On dependencies,* $(\bar{\boldsymbol{\nu}}x)\varepsilon$ *is the logical homomorphism such that:*

$$(\bar{\boldsymbol{\nu}}x)p_k \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \mathrm{n}(p) = x \text{ and } k = \mathbf{A} \\ \top & \text{if } \mathrm{n}(p) = x \text{ and } k = \mathbf{R} \\ p_k & \text{if } \mathrm{n}(p) \neq x \end{cases}$$

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

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

and

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

The degenerated case where $\{i \in I : \mathrm{n}(p_i) \neq x\}$ *is empty gives just* $\top$.

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

$$\begin{aligned} \big(a : \mathsf{Bool}, t : \lambda, f : \lambda; \\ (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}}) \triangleleft (b_{\mathbf{AR}} \wedge a_{\mathbf{AR}}) \vee \bar{f}_{\mathbf{A}} \triangleleft a_{\mathbf{AR}} \blacktriangleleft \\ a^{\omega} \wedge (t^1 \vee f^1)\big) \vdash \overline{a}(\boldsymbol{\nu} t' f').(t'.\bar{b}\langle tf\rangle + f'.\bar{f}) \end{aligned} \quad (40)$$

Subject Responsiveness. A port is responsive if it provides all resources given in the channel type, which is what the last statement in the (R-Pre) rule states. For $r(tf)$, this is written $r_{\mathbf{R}} \triangleleft (\bar{t}_{\mathbf{A}} \vee \bar{f}_{\mathbf{A}})$, where the right hand side is just (21) from page 30 with t and f replacing 1 and 2. Composing with (40) reduces the dependency chain and we obtain $r_{\mathbf{R}} \triangleleft (b_{\mathbf{AR}} \wedge a_{\mathbf{AR}})$, as required.

Replication. When typing a replicated process like $!\,a(\tilde{y}).P$ or $!\,(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle.P$, $\#(G)$ is ω and the last three factors must be replicated. We already saw in Definition 4.5.3 how to compute process type replication Γ^{ω}. We claim that Lemma 4.5.2 still holds when process types include dependency statements. We omit the proof but it can be shown in two parts: If Γ doesn't use disjunction then $n = 2$ satisfies the requirements (although $n = 1$ may also work for some processes) and Γ^2 replaces all non-zero multiplicities by $\star$. Secondly, if Γ'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.

6.7 Type System Tuning

In addition to what was noted in Section 4.6 for (R-Pre), (R-Sum) is not completely specified, as it permits some strengthening of Ξ_{E}. Again, the type

system is sound no matter how it is obtained the type system is sound, and we left it unspecified to permit some tuning of the type system behaviour. We suggest a few ways of choosing environment behaviour:

- For Ξ_{E} in (R-Sum), the simplest way is to just leave Ξ_{E} at the weakest possible permitted by the rule, but this is usually not desirable because it often causes ε, the sum activeness dependencies, to be $\bot$. On the other hand this permits deactivating the type system check for race-conditions like
$$(a.P{+}b.Q) \mid \bar{a} \mid \bar{b} \tag{41}$$
- A usually preferable way is to take $\Xi_{\mathrm{E}} = \bigwedge_i \Xi_{\mathrm{E}i} \wedge (\bigvee_i \bar{p}_i^{\star})$, which forces $(\sum_i p_i)_{\mathbf{A}}$ to hold, but would reject (41) as unsafe.

7 Further Extensions

7.1 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 (α depending on β and β on γ but α not depending on γ).

Consider the process $a(xy).\bar{x}.\bar{y} \mid \overline{a}\langle bc\rangle \mid b.c$, 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: $\bar{b}_{\mathbf{A}} \lhd a_{\mathbf{AR}}$. Reducing 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 (R-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 $\bar{t}.(a(x).b.x \mid \bar{b})$, 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

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

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

$\overline{a}\langle b\rangle$ 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 $z_{\mathbf{A}} \triangleleft \bar{q}_{\mathbf{A}} \triangleleft p_{\mathbf{R}}$, and after q has been consumed, we have $z_{\mathbf{A}}$ without dependencies and $\bar{q}_{\mathbf{A}} \triangleleft p_{\mathbf{R}} \triangleleft \bar{a}_{\mathbf{R}} \triangleleft \bar{b}_{\mathbf{A}}$.

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 Definition 5.0.1 on page 23 is extended as follows:

$$\Xi \;::=\; \cdots \;\;\big|\;\; l \;\;\big|\;\; \bar{l}$$

where l is taken from some infinite set disjoint from $\mathcal{N}$.

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 simplifying the events) 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.

The rule (R-Pre) is modified as follows:

$$\frac{\begin{array}{c}\Gamma \vdash P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x} \\ (\#(G) = 1 \text{ and } m' = \star) \Rightarrow \varepsilon = \bot\end{array}}{\begin{array}{rl}\Big(p : \sigma;\ \blacktriangleleft\, p^{m} \wedge \bar{p}^{m'}\Big) & \odot \\ \Big(;\, p_{\mathbf{A}}^{\#(G)} \triangleleft \varepsilon \blacktriangleleft\Big) & \odot \\ (\boldsymbol{\nu}\mathrm{bn}(G))\Big(\ \Gamma \triangleleft (l \vee \bar{p}_{\mathbf{A}}) & \odot \\ \overline{\sigma}[\tilde{x}] \triangleleft (l \vee \bar{p}_{\mathbf{AR}}) & \odot \\ \big(;\, p_{\mathbf{R}} \triangleleft (\bar{l} \vee \sigma[\tilde{x}])\ \blacktriangleleft\ \big)\Big)^{\#(G)} & \vdash G.P\end{array}} \text{ (R-Pre)}$$

7.2 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 (42) on page 61 where $\mathsf{Geom}_{\mathbf{R}} \triangleleft \overline{succ}_{\mathbf{R}}$ and $\overline{succ}_{\mathbf{R}} \triangleleft \mathsf{Geom}_{\mathbf{R}}$ reduce to $\mathsf{Geom}_{\mathbf{R}} \triangleleft \top$ rather than $\mathsf{Geom}_{\mathbf{R}} \triangleleft \bot$. This extension can of course be applied simultaneously to the previous one since they operate on different parts of the theory.

In a statement $\gamma \triangleleft \varepsilon$, a resource α in ε is now annotated with a *delay* α^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 $(p^m_{\mathbf{A}} \triangleleft \varepsilon) = p^m \wedge (p_{\mathbf{A}} \triangleleft \varepsilon)$ should probably be avoided as it would create confusion.

The following examples illustrate nicely the semantics of delays.

- $a_{\mathbf{R}} \triangleleft {u_{\mathbf{A}}}^2 \vdash a(x).\overline{x}(\boldsymbol{\nu} y).\bar{u}.\bar{y}$ — the u-dependency is only required after *two* exchanges on a (specifically, a and x)
- $\bar{u}_{\mathbf{A}} \triangleleft {\bar{a}_{\mathbf{AR}}}^{-2} \vdash a(x).\overline{x}(\boldsymbol{\nu} y).\bar{u}.\bar{y}$ — 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.
- ${b_{\mathbf{A}}} \triangleleft {a_{\mathbf{A}}}^0 \vdash \bar{a}.b$ — the delay is 0 because $a_{\mathbf{A}}$ is needed before one can even start interacting with b.
- ${a_{\mathbf{R}}} \triangleleft {b_{\mathbf{AR}}}^0 \vdash !\,a(x).\bar{b}\langle x\rangle$ — a (depth 1) answer from a depends on a (depth 1) answer from b.
- ${a_{\mathbf{R}}} \triangleleft {b_{\mathbf{AR}}}^2 \vdash !\,a(x).\overline{x}\langle b\rangle$ — in order to do n steps of a conversation with a, one needs to be able to do $n-2$ steps of a conversation 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 $\alpha \triangleleft \alpha^d$, α^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 ${a_{\mathbf{R}}} \triangleleft {b_{\mathbf{AR}}}^2 \odot {b_{\mathbf{R}}} \triangleleft {c_{\mathbf{AR}}}^2$, which reduces to ${a_{\mathbf{R}}} \triangleleft {c_{\mathbf{AR}}}^{2+2} = {a_{\mathbf{R}}} \triangleleft {c_{\mathbf{AR}}}^4$. If $c = a$ then we get $a_{\mathbf{R}} \triangleleft \top$.

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:

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

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 ${a_{\mathbf{R}}} \triangleleft {u_{\mathbf{A}}}^1$ becomes $\bar{t}_{\mathbf{A}} \triangleleft {u_{\mathbf{A}}}^{1-1} = \bar{t}_{\mathbf{A}} \triangleleft {u_{\mathbf{A}}}^0$.

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

$$\frac{\begin{array}{c}\Gamma \vdash P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x} \\ (\#(G) = 1 \text{ and } m' = \star) \Rightarrow \varepsilon = \bot\end{array}}{\begin{array}{rl}(\boldsymbol{\nu}\mathrm{bn}(G))\Big(\Big(p : \sigma;\ \blacktriangleleft\, p^{m} \wedge \bar{p}^{m'}\Big) & \odot \\ \big(;p^{\#(G)} \wedge (p_{\mathbf{A}} \triangleleft \varepsilon) \blacktriangleleft\big) & \odot \\ \Gamma \triangleleft \bar{p}_{\mathbf{A}}{}^{0} & \odot \\ \overline{\sigma}[\tilde{x}] \triangleleft \bar{p}_{\mathbf{AR}}{}^{-1} & \odot \\ \big(;p_{\mathbf{R}} \triangleleft \sigma[\tilde{x}]^{+1} \blacktriangleleft\big)\Big)^{\#(G)} & \vdash G.P\end{array}}\ (\text{R-PRE})$$

7.3 Properties

This section summarises the properties enjoyed by the type system.

Proposition 7.3.1 (Subject Congruence) *Let* $\Gamma \vdash P \equiv P'$. *Then* $\Gamma' \vdash P'$ *for some* $\Gamma' \cong \Gamma$.

Proposition 7.3.2 (Subject Reduction) *Let* $(\Gamma; P)$ *be a typed process such that* $\Gamma \vdash P$. *Then, for any transition* $(\Gamma; P) \xrightarrow{\mu} (\Gamma \wr \mu; P')$, $\exists \Gamma'$ *s.t.* $\Gamma' \preceq \Gamma \wr \mu$ *and* $\Gamma' \vdash P'$.

The proof is given in Section A.3.

Proposition 7.3.3 (Decidability) *There is a decidable algorithm that, given a channel type mapping* Σ *and a process* P, *either constructs a process type* $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ *where* $\Gamma \vdash P$ *and* Γ *is in normal form, or, if there is no such* Γ, *rejects the process.*

Proof This result follows from the type system being syntax directed and all operators being themselves decidable. Most operators have declarative definitions, in particular, logical homomorphisms are inductively defined on the behavioural statement structure, with the notable exceptions of replication (Definition 4.5.3) and closure (Definition 6.3.4). An algorithm for the former is given after the corresponding Lemma (page 21), and the proof of Lemma 6.3.5 includes an algorithm for computing closure. Finally, construction of a normal form is given in the proof of Lemma 6.3.2. □

And finally our main result, connecting decidable typability and undecidable semantics:

Proposition 7.3.4 (Type Soundness) *If* $\Gamma \vdash P$ *then* $\Gamma \models P$.

The proof is given in Section A.5.

8 Further Reading

In this section we present some related research, together with, when applicable, an encoding of their notation into ours, to help comparison.

8.1 Activeness

8.1.1 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 π-calculus with sums and matching (which we don't handle). A *linear receptive* name corresponds, in our terminology, to bi-linear names that are input active, like a in $a^{1}_{\mathbf{A}} \wedge \bar{a}^{1}$, and an *$\omega$-receptive* name is the same, but with ω 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 our process types by having a name a's local multiplicities be $\bar{a}^{\Gamma(a)} \wedge a^{\Delta(a)}$ for the linear type system (with $A(a) = 1$ if $a \in A$ and 0 otherwise), and the complement multiplicities $\bar{a}^{1-\Gamma(a)} \wedge a^{1-\Delta(a)}$ on the remote side. For the ω-receptiveness type system, we have, for each a, $\bar{a}^{\star\Gamma(a)} \wedge a^{\omega\Delta(a)}$ on the local side, and $\bar{a}^{\star} \wedge a^{\omega(1-\Delta(a))}$ on the remote one. Sangiorgi's *plain names* correspond to $a^{\star} \wedge \bar{a}^{\star}$, 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 ours. If however we weaken his soundness theorem to allow a weak input transition when using a receptive name, then our semantic definition matches his, and typability of our 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.

8.1.2 Pierce, Sangiorgi: Typing and Subtyping for Mobile Processes

This paper [PS93] studies input and output capabilities (in our terminology, types such as $\top$, $a^{\star}$, $\bar{a}^{\star}$, and $a^{\star} \wedge \bar{a}^{\star}$), and establishes a *subtyping* relation, which permits typing $\bar{a}\langle x\rangle$ 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 our notation, as follows:

$$[\![\, a : (\tilde{S})^{I} \,]\!] \stackrel{\text{def}}{=} \left(a : ([\![\, \tilde{S} \,]\!]); a^{\star I_{\mathbf{r}}} \bar{a}^{\star I_{\mathbf{w}}} \blacktriangleleft a^{\star \bar{I}_{\mathbf{r}}} \bar{a}^{\star \bar{I}_{\mathbf{w}}}\right)$$

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 ours, thanks to subtyping.

8.1.3 Kobayashi, Pierce, Turner: Linearity and the π-calculus

That paper [KPT99] is a specialisation of our 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 ω), and no behavioural

property. They also introduce $(\omega;\star)$ channels in section 7.3 (and call them $*$). Like in Section 8.1.2, we can encode their types as follows:

$$[\![\, a : p^m[\tilde{T}] \,]\!] \stackrel{\text{def}}{=} \left(a : ([\![\, \tilde{T} \,]\!]); a^{[\![\, m \,]\!]_{p_i}} \bar{a}^{[\![\, m \,]\!]_{p_o}} \blacktriangleleft a^{[\![\, m \,]\!]_{\bar{p}_i}} \bar{a}^{[\![\, m \,]\!]_{\bar{p}_o}}\right)$$

where mp_c is m if $c \in p$, 0 otherwise, $[\![\, 1 \,]\!] \stackrel{\text{def}}{=} 1$, 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.

8.1.4 Amadio et al.: The Receptive Distributed π-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 our terminology) active non-uniform ω 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 our type system without modification, as the closest we have is *uniform* activeness obtained through replication.

8.1.5 Acciai, Boreale: Responsiveness in process calculi

This paper [AB08a] addresses concerns very close to ours, through two distinct type systems. Note that what they call "responsiveness" is closer to what we call "activeness". Again, their setting is simpler than ours, in that it works on synchronous π, 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 our type system, in that it permits handling unbounded recursion such as a function computing the factorial of its parameter: $\mathbf{!}\, f(n,r).\,\mathrm{if}(n=0)\,\overline{r}\langle 1\rangle$ else $(\boldsymbol{\nu} r')\,(\overline{f}\langle n-1, r'\rangle \mid r'(m).\overline{r}\langle n * m\rangle)$. Our type system rejects such a process, because the recursive call would create a dependency $f_{\mathbf{R}} \triangleleft f_{\mathbf{R}}$.

We conjecture that their analysis, based on the well-foundedness of parameter domains, could be adapted to our behavioural statements by using *delays* (Section 7.2) typing $\mathbf{!}\, a(\tilde{y}).\bar{b}\langle\tilde{x}\rangle$ 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*, *ω-receptive* or *+-responsive*. For the last case they use a concept mostly equivalent to our multiplicities, which they call "capabilities". Their channel types can then be encoded into ours as follows:

- Inert type: $[\![\, a : I \,]\!] = \big(a : \lambda;\ \blacktriangleleft\ a^0 \wedge \bar{a}^0\big)$
- 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: $[\![\, 1 : T^{[\rho,k]} \,]\!] = (1 : ([\![\, 1 : T \,]\!]); \bar{a}_{\mathbf{A}} \blacktriangleleft a_{\mathbf{A}})$
- ω-receptive name: $[\![\, a : T^{[\omega,k]} \,]\!] = \big(a : ([\![\, 1 : T \,]\!]); a^{\omega}_{\mathbf{A}} \wedge \bar{a}^{\star} \blacktriangleleft a^0\big)$
- ω-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 ω, and on outputs, n, s, m and p correspond respectively to total multiplicities 0, $\star$, $\star$ and ω.

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 our "activeness" property, on processes in which responsiveness (in our 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 our 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 $\overline{a}\langle b\rangle$ 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

$$!\, a(x).\bar{x} \,|\, !\, a(x).\overline{a}(\boldsymbol{\nu} y).y.\overline{a}\langle x\rangle \,|\, \overline{a}\langle s\rangle$$

implementing a random walk. 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 ω-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 ours.

Similarly to what was presented in this paper, their first type system uses dependency analysis to check strong linear activeness or strong ω-activeness on input ports, and activeness for linear output ports. For a process like $\bar{b} \mid b.\bar{a}$, 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 our *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 our type system, such as "half-linear names" like t in $(\boldsymbol{\nu} t)\,(\bar{t} \mid t.P \mid t.Q)$, as well as processes such as $(\boldsymbol{\nu} a)\,(a(x).(\bar{x} \mid b(y).\bar{y}) | \overline{a}\langle t\rangle)$ 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 our 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 ours 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: $b(x).\overline{a}\langle x\rangle \mid c(x).a(y).\overline{x}\langle y\rangle \mid \overline{c}\langle b\rangle$, where all names are assumed responsive (in their terminology, or "bi-linear active" in ours). That process should be ruled out because it reduces to $b(x).\overline{a}\langle x\rangle \mid a(y).\bar{b}\langle y\rangle$, 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 $b_{\mathbf{R}} \lhd \bar{c}_{\mathbf{R}} \lhd a_{\mathbf{R}} \lhd b_{\mathbf{R}}$ 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 our 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.

8.1.6 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 8.1.1 (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 8.1.2 (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 $P = !\,a(x).\bar{b}\langle x\rangle \mid !\,b(x).\overline{a}\langle x\rangle \mid \overline{a}\langle s\rangle \mid s$, 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 our multiplicities, and tell for a particular channel how many times the input and output ports are used, and in what order.

Definition 8.1.3 (Channel Usages) *The* usage *of a channel is an expression given by the following grammar:*

$$U \;::=\; \mathbf{0} \;\mid\; \rho \;\mid\; u.U \;\mid\; (U|U) \;\mid\; U\&U \;\mid\; \mu\rho.U$$
$$u \;::=\; ! \;\mid\; ?$$

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, $a.b \mid \bar{b}.\bar{c}$ uses a according to $?$, b according to $?|!$ and c according to $!$. In $!\,a(x).\overline{x}\langle 1\rangle$, a has usage $*?|!$ (with $*? \stackrel{\text{def}}{=} \mu\rho.(?.\rho)$), 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 $(\boldsymbol{\nu} t)\,(\bar{t} \mid t.P \mid t.Q)$.

Obligation and Capability levels generalise the levels used in [AB08a]:

Definition 8.1.4 (Obligation and Capability Levels) *An* obligation level *for an (input or output) primitive is a number (or ∞) 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 $u \;::=\; !^{t_O}_{t_C} \;\mid\; ?^{t_O}_{t_C}$.

For example, consider the process $a.b \mid \bar{b}.\bar{c}$. 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 our 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 ω-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, it uses numerical levels, which permit forcing basic dependency relations between elements of the usages of different channels, but can't encode selection or branching as it amounts to having no "$\vee$" in behavioural statements. Moreover, events described in Section 7.1 permit an accurate analysis of processes such as

$$(\boldsymbol{\nu} t)\left(\bar{t} \,\middle|\, t.(!\, z | !\, a(x).\bar{z}.\bar{x}) \,\middle|\, t.!\, a(y).\bar{y}\right)$$

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 our 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 our system.

Our 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.

8.1.7 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 8.1.6 therefore remains mostly valid. The paper 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.

8.1.8 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 ours, and their "robust" properties are analogous to our 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.

8.2 Generic Type Systems

The three following papers have a *generic* approach, as opposed to the previous ones (and the present paper) that are aimed at specific properties. They have to be *instantiated* with the desired property, expressed in various ways.

8.2.1 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 (in contrast with activeness with is a *liveness* property) 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. Types use "+" in essentially the same sense as we do, and "&" corresponds precisely to our $\vee$. The paper includes as examples of instantiations, simple arity-mismatch checking system, race-freedom and deadlock-freedom type systems.

8.2.2 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 our $\triangleleft$ connective, so conditional properties need to be encoded by modifying the activeness formulæ, which may become very complex for statements such as (34) 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 ours, 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 our type system due to over-approximation.

8.2.3 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.

8.3 Structural Analysis

8.3.1 Bodei, Degano et al: Control Flow Analysis for the π-calculus

Related to the *structural analysis* used for the soundness proof (Section A.5), 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) π-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 α-renaming by inserting channel and binder markers into the process syntax, then referring to channels by channel markers rather than names, rather similarly to our "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 our 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. Our 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 used 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 γ 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).

9 Conclusion

We conclude this paper by reviewing its contributions, the power and limitations of the type system, and possible future research directions.

We described a type notation and semantics that combine statements about liveness properties ($s_{\mathbf{A}}$ and $p_{\mathbf{R}}$), choice (through branching $(p{+}q)_{\mathbf{A}}$ and selection $\varepsilon\vee\varepsilon'$) and conditional properties ($\varepsilon\triangleleft\varepsilon'$). Then the type system outlined in section 6 is able, given a process P, channel types and optionally port multiplicities, to construct a process type whose local component Ξ_{L} contains all information the type system was able to gather about P's behaviour. As the type system is sound and decidable, it is necessarily incomplete, but still powerful enough to recognise activeness and responsiveness in many important applications such as

data representation (we extensively covered an encoding of Boolean values and operators) or object-oriented style programming. We give an example of the latter:

An *object* o responding to methods $m_1, \ldots, m_n$ is represented by a replicated input

$$!\, o(m_1 \ldots m_n).\big(m_1(\tilde{y}).P_1 + \cdots + m_n(\tilde{y}).P_n\big)$$

and calling o's method m_i is then done as follows.

$$\overline{o}(\boldsymbol{\nu} m_1 \ldots m_n).\overline{m_i}\langle \tilde{x} \rangle$$

Then, when passing an object o around, one can require the sender to be input active and responsive, meaning that o must be ready to receive a method call, and the receiver must be output responsive, meaning that any method call must obey o's protocol (an output at o must be followed by an output at exactly one of its parameters).

Sessions are another example where responsiveness and choice appear together. Our types require a channel's type not to evolve over time but this issue is avoided by having as many channel types as the session contains states, and passing, the next state as an additional parameter at each step.

Regarding the limitations of our work, we chose to focus on choice itself, leaving out features like recursivity [AB08a] or subtyping [PS93], and complex channel usages such as locks [Kob02a], which have been well explored before in a choice-less context. Note however that integrating recursivity would be non-trivial because nothing in an encoded Integer type prevents numbers to be infinite, and yet it may be desirable in some contexts to permit unbounded numbers. For instance

$$!\, \mathsf{Geom}(zero, succ).(\overline{zero} + \overline{succ}\langle \mathsf{Geom} \rangle) \tag{42}$$

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 "**F**" 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.

$$r^{\omega}_{\mathbf{A}} \;\wedge\; r_{\mathbf{R}} \triangleleft (a_{\mathbf{AF}} \wedge b_{\mathbf{AR}}) \;\wedge\; r_{\mathbf{F}} \triangleleft (a_{\mathbf{AF}} \wedge b_{\mathbf{AF}}) \vdash$$
$$!\, r(zs).(\boldsymbol{\nu} t)\,(\overline{t}\langle a \rangle \mid !\, t(x).\overline{x}(\boldsymbol{\nu} z's').(z'.\overline{b}\langle zs \rangle + s'(x').\overline{t}\langle x' \rangle))$$

Note that our current type system (with the "delayed dependencies" extension) recognises that Geom is responsive, but due to t calling itself, just produces $r_{\mathbf{R}} \triangleleft \bot$.

We are currently working on a generic form of the type system exposed in this paper, that captures the essence of dependency analysis, and can be *instantiated* with various liveness and safety properties such as **A** and **R** but also others such as termination and determinacy.

Acknowledgements

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).

References

[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 π-calculus. *ACM Transactions on Programming Languages and Systems*, 25(5):549–577, 2003.

[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.

[Cai04] L. Caires. Behavioral and Spatial Observations in a Logic for the π-Calculus. In *Proceedings of FOSSACS'04*, volume 2987 of *LNCS*. Springer, 2004.

[DS06] Y. Deng and D. Sangiorgi. Ensuring termination by typability. *Information and Computation*, 204(7):1045–1082, 2006.

[GR09] M. Gamboni and A. Ravara. Activeness and Responsiveness in Mobile Processes. In *7th Conference on Telecommunications*, pages 429–432. Instituto de Telecomunicações, 2009.

[GR10] M. Gamboni and A. Ravara. Responsive Choice in Mobile Processes. In M. Wirsing, M. Hofmann and A. Rauschmayer, eds, *Trustworthy Global Computing*, volume 6084 of *LNCS*, pages 135–152. Springer, Heidelberg, 2010.

[HY02] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Symposium on Principles of Programming Languages*, pages 81–92. ACM, 2002.

[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.

[Mil93] R. Milner. The Polyadic π-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.

[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.

[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.

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 Algebraic Properties

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 ε 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 6.3.1 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 $!\,(a+b)$ (but includes the strongly bisimilar $!\,a|!\,b$).

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:*

$$P \equiv (\boldsymbol{\nu}\tilde{z})\ \left(Q \mid G.R\right)$$

where $\mathrm{n}(p) \notin \tilde{z}$ and $\mathsf{sub}(G) = p$, and, if μ is an output, $\mathsf{obj}(G) \cap (\mathrm{bn}(G) \cup \tilde{z}) = \mathrm{bn}(\mu)$. For P', either (when $\#(G) = 1$)

$$P' \equiv (\boldsymbol{\nu}\tilde{z} \setminus \mathrm{bn}(\mu))\ \left(Q \mid R\{{}^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}\right)$$

or (when $\#(G) = \omega$)

$$P' \equiv (\boldsymbol{\nu}\tilde{z} \setminus \mathrm{bn}(\mu))\ \left(Q \mid G.R \mid R\{{}^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}\right).$$

Now let instead $P \xrightarrow{\tau} P'$, still not using (Sum)*. Then*

$$P \equiv (\boldsymbol{\nu}\tilde{z})\ \left(Q \mid G.R \mid G'.R'\right)$$

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:

$$P' \equiv (\boldsymbol{\nu}\tilde{z} \cup \mathrm{bn}(G'))\ \left(Q \mid \left(R\{{}^{\mathsf{obj}(G')}/_{\mathsf{obj}(G)}\}\right) \mid R'\right).$$

Finally, the following lemma gives a few useful properties of process type operators:

Lemma A.1.5 *Let Γ_1 and Γ_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 Γ_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 5.1.2)

> 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 Normal Form (Lemma 6.3.3)

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

$$\bigvee_{j\in J} \varepsilon'_j \preceq \bigvee_{j\in J} \varepsilon_j \tag{43}$$

By idempotence, multiple ε'_j equal to the same ε_i can be replaced by a single one, so we have

$$\bigvee_{i\in I_0} \varepsilon_i \cong \bigvee_{j\in J} \varepsilon'_j \tag{44}$$

where $I_0 = \{i \in I \ : \ \exists j \in J : \varepsilon'_j = \varepsilon_i\}$. Applying the $\varepsilon \vee \varepsilon' \preceq \varepsilon$ rule we get

$$\bigvee_{i\in I} \varepsilon_i \preceq \bigvee_{i\in I_0} \varepsilon_i \tag{45}$$

as $I_0 \subseteq I$. Composing the three above relations we have the desired inequality.

A.1.4 Closure Uniqueness (Lemma 6.3.5)

We proceed in increasing generality, by first focusing on special cases. Let:

$$\Delta = \bigwedge_{i\in I} \gamma_i \triangleleft \varepsilon_i \tag{46}$$

where $\gamma_i \neq \gamma_{i'}$ for any distinct i and i'. We only consider points 1, 2 and 4 from Definition 6.3.4 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 ε 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 $(\bigwedge_{i\in I:\gamma_i\notin\tilde{\alpha}} \gamma_i \triangleleft \varepsilon_i) \wedge (\bigwedge_{\alpha\in\tilde{\alpha}} \alpha \triangleleft \bot)$, and $\hat{\Delta}(\gamma_i)$ is $\hat{\varepsilon}_i$, γ_i's dependencies in $\hat{\Delta}$. The following definition can be used to construct a closure explicitly:

Definition A.1.7 (Δ-Closure) *A* Δ-closure *of a statement* Θ *(typically chosen equal to* Δ*) is a statement* $\mathrm{close}_{(\Delta)}(\Theta) = \Theta'$ *inductively constructed as follows:*

1. $\mathrm{close}_{(\Delta)}(\top) \stackrel{\text{def}}{=} \top$ *and* $\mathrm{close}_{(\Delta)}(\bot) \stackrel{\text{def}}{=} \bot$
2. $\mathrm{close}_{(\Delta)}(\gamma \triangleleft \varepsilon) \stackrel{\text{def}}{=} \gamma \triangleleft (\mathrm{close}_{(\Delta \setminus \gamma)}(\varepsilon))$
3. $\mathrm{close}_{(\Delta)}(\gamma) \stackrel{\text{def}}{=} \gamma * \mathrm{close}_{(\Delta \setminus \gamma)}(\Delta(\varepsilon))$.
4. $\mathrm{close}_{(\bigwedge_{i \in I} \gamma_i \triangleleft \varepsilon_i)}(\gamma) \stackrel{\text{def}}{=} \gamma$ *if* $\nexists i \in I : \gamma_i = \gamma$.
5. $\mathrm{close}_{(\Delta)}(\Delta_1 \wedge \Delta_2) \stackrel{\text{def}}{=} \mathrm{close}_{(\Delta)}(\Delta_1) \wedge \mathrm{close}_{(\Delta)}(\Delta_2)$.

It is easily seen by induction on the number of symbols appearing in the representation of Δ plus the number of statements in Θ that do not depend on ⊥, 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 Δ' such that $\hat{\Delta} \hookrightarrow \Delta'$ satisfies $\hat{\Delta} \cong \Delta'$. In other words, for all distinct j and k:

$$\varepsilon'_k \stackrel{\text{def}}{=} \hat{\varepsilon}_k\{^{\gamma_j * (\hat{\varepsilon}_j\{^{\bot}/_{\gamma_k}\})}/_{\gamma_j}\} \cong \hat{\varepsilon}_k \tag{47}$$

By construction, every γ_i appearing on the rhs of a $\triangleleft$ 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 γ_i). Moreover, within a statement $\gamma_i \triangleleft \varepsilon$ or $\gamma_i * \varepsilon$, any γ_i appearing in ε occurs as $\gamma_i * \bot$.

Assume w.l.o.g. that γ_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 (47) we get

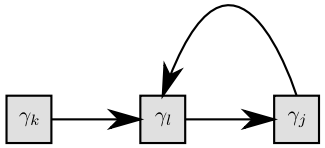
$$\varepsilon'_k = C_{\mathbf{k}}[\gamma_j * \big(\mathrm{close}_{(\Delta \setminus \tilde{\gamma})}(\varepsilon_j) \ , \ \mathrm{close}_{(\Delta \setminus \{\gamma_j, \gamma_k\})}(\varepsilon_j)\big)] \tag{48}$$

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 γ_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

$$\hat{\varepsilon}_j = C_{\mathbf{j}}[\gamma_l * \mathrm{close}_{(\Delta \setminus \tilde{\gamma}')}(\varepsilon_l)] \tag{49}$$

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 γ_k to γ_j, and $C_{\mathbf{j}}[\cdot]$ is represented by the arrow going back from γ_j to γ_l.



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 $\mathrm{close}_{(\Delta \setminus \tilde{\gamma}')}(\varepsilon_l) = C_{\mathbf{l}}[\gamma_j * \bot]$ (note that $\gamma_j \in \tilde{\gamma}'$ implies $\tilde{\gamma}'(\gamma_j) = \bot$).

Composing (47) and (49) 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 $\Delta' = \mathrm{close}_{(\Delta)}(\Delta)$ is a closure. We still need to show that it is the only closure, i.e. any closure of Δ is $\cong$-equivalent to Δ'.

Let $\Delta \hookrightarrow \Delta''$ be s.t. $\Delta'' \hookrightarrow \Delta'''$ implies $\Delta'' \cong \Delta'''$ for all Δ'''.

By the definition of $\hookrightarrow$, Δ'' can be obtained from Δ by, a certain number of times, replacing γ_i by $\gamma_i * \varepsilon_i$. (Technically an individual application of a rule in 6.3.4 introduces some ε'_i not necessarily equal to ε_i but as ε'_i was itself obtained from ε_i by applying similar transformations, this description is correct).

A resource occurrence γ_j in a statement is said "bare" if it is neither followed by the $*$-operator nor contained in the ε of a statement $\gamma_j * \varepsilon$.

A bare occurrences of a resource γ_j can be "completed" by applying Definition A.1.6 to replace all γ_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 Δ''' that has no bare resource occurrences, and that satisfies $\Delta'' \hookrightarrow \Delta'''$. As Δ'' was assumed to be a closure, $\Delta'' \cong \Delta'''$. Nested resource developments ($\gamma_i * \varepsilon$ where ε contains $\gamma_i * \varepsilon'$ for some ε' can be reduced as shown above (replacing $\gamma_i * \varepsilon'$ by $\gamma_i * \bot$), resulting in $\Delta''' \cong \mathrm{close}_{(\Delta)}(\Delta)$, as required.

A.1.5 Composition of Disjoint Statements (Lemma 6.3.10)

According to Convention 6.2.2, Ξ and Ξ' 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 Ξ', $\{p_i\}_{i \in R}$ is the set of ports whose responsiveness appear in Ξ' on the lhs of a dependency "$\triangleleft$" (and the other way round for M' and R').

In other words,

$$\Xi \odot \Xi' \stackrel{\mathrm{def}}{=} \Xi_r = \big(\Xi \wedge \bigwedge_{i \in M} {p_i}^0 \wedge \bigwedge_{i \in R} p_{i\mathbf{R}} \triangleleft \top\big) \odot \big(\Xi' \wedge \bigwedge_{i \in M'} {p_i}^0 \wedge \bigwedge_{i \in R'} p_{i\mathbf{R}} \triangleleft \top\big). \quad (50)$$

From the Ξ_r written in (50) onwards, until the end of this proof, Convention 6.2.2 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,

$$\Xi_r \cong (\Xi \odot \Xi') \wedge \big((\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_{i\mathbf{R}}{\triangleleft}\top) \odot \Xi'\big) \wedge \big(\Xi \odot (\bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_{i\mathbf{R}}{\triangleleft}\top)\big) \wedge \big((\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_{i\mathbf{R}}{\triangleleft}\top) \odot (\bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_{i\mathbf{R}}{\triangleleft}\top)\big) \quad (51)$$

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 Ξ and Ξ' have no common resources and M, R, M' and R' index resources in Ξ and Ξ', 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 Ξ' using $\odot$'s logical homomorphism. We obtain a behavioural statement equal to Ξ' where every atomic statement p^m or $\gamma{\triangleleft}\varepsilon$ got replaced by $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_{i\mathbf{R}}{\triangleleft}\top) \odot p^m$ or $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_{i\mathbf{R}}{\triangleleft}\top) \odot (\gamma{\triangleleft}\varepsilon)$, respectively. In the first case, $\exists i \in M$ s.t. $p_i = p$, so it is equal to

$$\bigwedge_{i\in M;\, p_i\neq p} ({p_i}^0 \odot p^m) \;\wedge\; (p^0 \odot p^m) \;\wedge\; \bigwedge_{i\in R} (p_{i\mathbf{R}}{\triangleleft}\top \odot p^m)$$

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

$$\bigwedge_{i\in M} ({p_i}^0 \odot (\gamma{\triangleleft}\varepsilon)) \;\wedge \bigwedge_{i\in R;\, p_{i\mathbf{R}}\neq\gamma} (p_{i\mathbf{R}}{\triangleleft}\top \odot (\gamma{\triangleleft}\varepsilon)) \;\wedge\; (\gamma{\triangleleft}\top \odot (\gamma{\triangleleft}\varepsilon))$$

i.e. (using point 3 of the Definition on the right, and 5 for the rest)

$$\bigwedge_{i\in M} \top \;\wedge \bigwedge_{i\in R;\, p_{i\mathbf{R}}\neq\gamma} \top \;\wedge\; ((\gamma{\triangleleft}\top) \wedge (\gamma{\triangleleft}\varepsilon)) \cong \gamma{\triangleleft}\varepsilon$$

Finally, for an activeness statement, noting that $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_{i\mathbf{R}}{\triangleleft}\top) \cong (\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_{i\mathbf{R}}{\triangleleft}\top) \wedge \top \cong (\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_{i\mathbf{R}}{\triangleleft}\top) \wedge (\gamma{\triangleleft}\bot)$:

$$\bigwedge_{i\in M} ({p_i}^0 \odot (\gamma{\triangleleft}\varepsilon)) \;\wedge\; \bigwedge_{i\in R} (p_{i\mathbf{R}}{\triangleleft}\top \odot (\gamma{\triangleleft}\varepsilon)) \;\wedge\; (\gamma{\triangleleft}\bot \odot (\gamma{\triangleleft}\varepsilon))$$

i.e. (using point 2 of the Definition on the right, and 5 for the rest)

$$\bigwedge_{i\in M} \top \;\wedge \bigwedge_{i\in R;\, p_{i\mathbf{R}}\neq\gamma} \top \;\wedge\; ((\gamma{\triangleleft}\bot) \vee (\gamma{\triangleleft}\varepsilon)) \cong \gamma{\triangleleft}\varepsilon$$

We conclude that $(\bigwedge_{i\in M} {p_i}^0 \wedge \bigwedge_{i\in R} p_{i\mathbf{R}} \triangleleft \top) \odot \Xi' \cong \Xi'$, and similarly $\Xi \odot (\bigwedge_{i\in M'} {p_i}^0 \wedge \bigwedge_{i\in R'} p_{i\mathbf{R}} \triangleleft \top) \cong \Xi$, so (51) becomes $\Xi_r \cong \top \wedge \Xi' \wedge \Xi \wedge \top \cong \Xi \wedge \Xi'$ and we're done.

A.1.6 Composition Properties (Lemma 4.3.8)

The + operator on multiplicities is commutative as can be seen in Definition 4.3.3. 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 6.3.1).

Commutativity of behavioural statement composition The $\Delta_1 \odot \Delta_2 \cong \Delta_2 \odot \Delta_1$ equivalence is proven by structural induction on Δ_1 and Δ_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) $(\Theta_1 \odot \Delta_2) \wedge (\Theta_2 \odot \Delta_2)$ which 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 6.3.8 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 Δ_i are of the form p^m and $\gamma \triangleleft \varepsilon$. Note that if they are not all dependency statements of the same resource γ, or all multiplicities of the same port p, rule 4 of Definition 6.3.8 will apply and return $\top$ no matter in which order the Δ_i are composed. Otherwise, the three remaining base cases corresponding to the first three points of Definition 6.3.8 satisfy associativity as a consequence of $+$, $\vee$ and $\wedge$ being associative up to $\cong$.

As a corollary of Lemma 6.3.10, $\top$ is a neutral element of $\odot$ when Convention 6.2.2 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,

$$(\Sigma_1 \wedge \Sigma_2 \,;\, \Xi_{\mathrm{L}1} \odot \Xi_{\mathrm{L}2} \blacktriangleleft (\Xi_{\mathrm{E}1} \setminus \Xi_{\mathrm{L}2}) \wedge (\Xi_{\mathrm{E}2} \setminus \Xi_{\mathrm{L}1})) \cong (\Sigma_2 \wedge \Sigma_1 \,;\, \Xi_{\mathrm{L}2} \odot \Xi_{\mathrm{L}1} \blacktriangleleft (\Xi_{\mathrm{E}2} \setminus \Xi_{\mathrm{L}1}) \wedge (\Xi_{\mathrm{E}1} \setminus \Xi_{\mathrm{L}2})) \quad (52)$$

As closure and removal of non-observable dependencies commute with $\cong$ (Lemma 5.1.3), $\odot$ on process types is commutative.

$(\varnothing; \top \blacktriangleleft \top)$ is a neutral element: Let Δ be any behavioural statement. Then $\Delta \setminus \top = \Delta$, and $\top \setminus \Delta = \top$, both consequences of point 4 in Definition 5.1.6. 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 6.3.11 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 6.3.11) for computing Γ 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:

$$\forall \Delta, \Delta_1, \Delta_2 : (\Delta_1 \hookrightarrow \Delta_2) \;\Rightarrow\; (\Delta \setminus \Delta_1 \cong \Delta \setminus \Delta_2) \tag{53}$$

We omit the proof but essentially, dependency reduction preserves the only parts of Δ_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 5.1.6), 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 Γ 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.2 Semantic Properties

A.2.1 Simple Correctness and Structural Equivalence (L. 4.4.2)

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 4.4.1 is an immediate consequence of $\Gamma \models_{\#} P$ and $\equiv$ preserving the set of free names.

Point 2 of Definition 4.4.1 is an immediate consequence of $\Gamma \models_{\#} P$ and $\equiv$ being a bisimulation, keeping for Q the same Γ_+ that was used for P.

Point 3 of Definition 4.4.1 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 μ 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 4.4.1 is satisfied as the channel types in Γ' and Θ' 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 Γ_+, and let Θ_+ be equal to Θ' but setting environment multiplicities of $\tilde{p}$ to $\star$. Again, as $\cong$ commutes with $\wr$, keeping the same μ' as with Γ_+, $\Theta_+ \wr \mu'$ is well defined.

Property number 3 is satisfied because the multiplicity of a port is preserved by type equivalence.

A.2.2 Bisimulation and Type Equivalence (Lemma 6.4.5)

Inspecting Definition 6.4.4, it is clear that $\Gamma \models P$ is only concerned about transition sequences available from P, and not about P's structure (beyond the assumption that $\Gamma \models_{\#} P$ but this is assumed in Lemma 6.4.5 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 6.4.4. Let $\Gamma \preceq \Theta$. We show that $\Theta \models P$.

We use the following properties whose proofs are omitted:

1. Let $\Gamma \preceq \Theta$. If $\Gamma \wr \tilde{\mu} = \Gamma'$ then $\Gamma' \preceq (\Theta \wr \tilde{\mu})$.
2. For any statement Γ' with $\Gamma' \preceq \Theta'$, for any projection $\Gamma' \searrow \Gamma''$ there is a projection $\Theta' \searrow \Theta''$ such that $\Gamma'' \preceq \Theta''$.
3. Let $\Gamma'' \preceq \Theta''$ be two elementary statements, and $\Gamma'' \cong \bigvee_{j \in J} \gamma_j \triangleleft \varepsilon_j$. Then $\Theta'' \cong \bigvee_{j \in J'} \gamma_j \triangleleft \varepsilon'_j$ where $J \subseteq J'$ and $\forall j \in J : \varepsilon_j \succeq \varepsilon'_j$.

There exists thus a tight matching[8] between the transition network starting from $(\Gamma; P)$ and the one from $(\Theta; P)$, which permits translating f into a strategy

[8]Note that this matching need not be unique because an elementary statement can be weakened to a non-elementary statement, as in $\alpha \preceq \alpha \vee (\beta_1 \wedge \beta_2)$ which has two projections $\alpha \vee \beta_i$. However the proof works no matter which projection is chosen.

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 $(\Gamma; P)$. By construction, Then $f'(\Theta'; P')$ is the typed process corresponding to $f(\Gamma'; P')$.

Consider an infinite transition sequence, a strategy application set I, and and indexing set J^Θ (we put the Θ-annotation to make it clear which sequence is being talked about) and as in the Definition but starting with $(\Theta; P) \xrightarrow{\tilde{\mu}_q} (\Theta_0; P_0)$. Using the above defined mapping there is a corresponding transition sequence from $(\Gamma; P)$, which satisfies the requirements 2.1, 2.2 and 2.3 of the Definition with the same I and μ_i, and (following property 3 above) with an indexing set $J^\Gamma \subseteq J^\Theta$. The properties γ_j are the same for the two sequences but their dependencies satisfy

$$\forall i \in \mathbb{N}_0, j \in J^\Gamma : \varepsilon^\Gamma_{j,i} \succeq \varepsilon^\Theta_{j,i} \tag{54}$$

As $\Gamma \models P$, there is $j \in J^\Gamma$ and (if 3.2 applies) $i \in I$ satisfying 3.1, 3.2 and 3.3, using $\varepsilon^\Gamma_{j,i}$ in place of $\varepsilon_{j,i}$.

The same j (and, if applicable, i) show that $\Theta \models P$ as well. For point 3.1, (54) gives $\varepsilon^\Theta_{j,i} \preceq \varepsilon^\Gamma_{j,i} \preceq \overline{p_i}_\mathbf{A}$, so $\varepsilon^\Theta_{j,i} \preceq \overline{p_i}_\mathbf{A}$ as well. For point 3.2, if immediate correctness of $\gamma_j \triangleleft \varepsilon^\Gamma_{j,i}$ relied on point 1 of Definition 6.4.1 then we must have $\varepsilon^\Gamma_{j,i} = \bot$, so by (54) $\varepsilon^\Theta_{j,i} = \bot$ as well, implying immediate correctness of $\gamma_j \triangleleft \varepsilon^\Theta_{j,i}$ as well. If immediate correctness of the former relied on point 6.4.1.2 then (noting that the definition doesn't refer to ε) it applies to the latter as well. Finally, if point 3.3 of Definition 6.4.4 relied on point 6.4.1.3, the only occurrence of $\varepsilon = \varepsilon^\Gamma_{j,i}$ in that rule being on the rhs of a "$\succeq$" relation, it remains true when $\varepsilon^\Gamma_{j,i}$ is replaced by the stronger $\varepsilon^\Theta_{j,i}$.

A.3 Subject Reduction

In this section we prove Proposition 7.3.2.

We show that, if $\Gamma \vdash P$, $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ and $\Gamma_0 \vdash P'$ then $\Gamma_0 \preceq \Gamma'$.

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 $\Gamma \vdash G.P$ and $\hat{\Gamma} \vdash G.P + Q$, the latter being obtained from the former using (R-Sum).

From (R-Sum), $\hat{\Gamma} = (\mathsf{sub}(G) + s)_\mathbf{A} \triangleleft \varepsilon \wedge (\Gamma \vee \Gamma_Q)$, for some ε, s and Γ_Q depending on S. By $\Xi_1 \vee \Xi_2 \succeq \Xi_1$, $\hat{\Gamma} \succeq \left(\sum_{i\in I} p_i\right)_\mathbf{A} \triangleleft \varepsilon \wedge \Gamma$.

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 μ.

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 τ-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.3.1 τ-Reductions

First assume $\mu = \tau$. Then, by Lemma A.1.4,

$$P \equiv (\boldsymbol{\nu} \tilde{z})\,(Q \mid \sum_{i \in I} G_i.P_i \mid \sum_{i' \in I'} G_{i'}.P_{i'}), \tag{55}$$

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.3.1 (The Sums are not Active) *Let $\Gamma \vdash P$ with P as in (55).*

Then Γ's local behavioural statement does not contain $(\sum_{i \in I} \mathsf{sub}(G_i))_{\mathbf{A}} \triangleleft \varepsilon$ or $(\sum_{i' \in I'} \mathsf{sub}(G_{i'}))_{\mathbf{A}} \triangleleft \varepsilon$ for $\varepsilon \not\cong \bot$.

Proof Let

$$(\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}}) \vdash \sum_{i \in I} G_i.P_i \quad \text{and} \quad (\Sigma'; \Xi'_{\mathrm{L}} \blacktriangleleft \Xi'_{\mathrm{E}}) \vdash \sum_{i' \in I'} G_{i'}.P_{i'}$$

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 Ξ_{E} and Ξ'_{L}. Then, assume Ξ_{L} contains $(\sum_{i \in I} \mathsf{sub}(G_i))_{\mathbf{A}}$ (if it doesn't, we're done). By (R-Sum), Ξ_{E} must have no concurrent $p_{i'}$:

$$\forall j \in J : \big(\Xi_j \setminus \bar{a} \cong \bot \quad \text{or} \quad \forall i \in I \setminus \{\hat{\imath}\} : \Xi_j \setminus \overline{\mathsf{sub}(G_i)} \cong \bot\big) \tag{56}$$

As $\mathsf{sub}(G_{\hat{\imath}'}) = \bar{a}$, there is $m \neq 0$ s.t. $\bar{a}^m \vdash G_{\hat{\imath}'}.P_{\hat{\imath}'}$, which gets carried over by (R-Sum) to Ξ'_{L} as

$$\bar{a}^m \succeq \Xi'_{\hat{k}} \tag{57}$$

for some $\hat{k}$. Now, when applying (R-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 5.1.6) 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 ρ and let $j = \rho(\hat{k})$. Then, by (56) and (57), either $\Xi_j \setminus \Xi'_{\hat{k}} \cong \bot$ (in case $\Xi_j \setminus \bar{a} \cong \bot$) 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 $\Xi \setminus p = \bot$ iff $p^0 \succeq \Xi$). All j in the first case drop from the disjunction over ρ. 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 Definition 6.3.6, at step three of Definition 6.3.11.

It can be similarly shown that P's behavioural statement doesn't contain $(\sum_{i \in I'} \mathsf{sub}(G_i))_{\mathbf{A}} \triangleleft \varepsilon'$ for $\varepsilon' \not\cong \bot$. □

Removing the sums from (55) we get

$$(\boldsymbol{\nu} \tilde{z})\,(Q \mid G_{\hat{\imath}}.P_{\hat{\imath}} \mid G_{\hat{\imath}'}.P_{\hat{\imath}'}) \tag{58}$$

Lemma A.3.1 implies that (55)'s type is stronger than (58)'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 $P = \bar{a}\langle\tilde{x}\rangle.O \mid a(\tilde{y}).I$, 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 $\Gamma_O \vdash O$ and $\Gamma_I \vdash I$. The input's type is $(\boldsymbol{\nu}\tilde{y})\,\Gamma'_I$ where, using (R-Pre),

$$\Gamma'_I = a : \sigma \odot a_{\mathbf{A}} \triangleleft \varepsilon \odot a_{\mathbf{R}} \triangleleft (\bar{l} \vee \sigma[\tilde{y}]) \odot \bar{\sigma}[\tilde{y}] \triangleleft (l \vee \bar{a}_{\mathbf{AR}}) \odot \Gamma_I \triangleleft (l \vee \bar{a}_{\mathbf{A}}) \tag{59}$$

and the output is typed as

$$\Gamma'_O = a : \sigma \odot \bar{a}_{\mathbf{A}} \triangleleft \varepsilon' \odot \bar{a}_{\mathbf{R}} \triangleleft (\bar{l}' \vee \bar{\sigma}[\tilde{x}]) \odot \sigma[\tilde{x}] \triangleleft (l' \vee a_{\mathbf{AR}}) \odot \Gamma_O \triangleleft (l' \vee a_{\mathbf{A}}). \tag{60}$$

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 Γ_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 Γ_I are

$$\bigwedge_{i\in\mathcal{Y}} \gamma_i \triangleleft \varepsilon_i \;\wedge\; \bigwedge_{i\in\mathcal{O}} \gamma_i \triangleleft \varepsilon_i \tag{61}$$

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 ε_i is sometimes written in the following *normal form* (which, although not unique, always exists):

$$\forall i \in \mathcal{O} \cup \mathcal{Y} : \varepsilon_i = \bigvee_{j\in\mathcal{I}_i} \varepsilon^O_{ij} \wedge \varepsilon^Y_{ij} \tag{62}$$

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:)

$$\sigma[\tilde{y}] = \bigwedge_{i\in\mathcal{Y}} \gamma_i \triangleleft \varepsilon^P_i \quad \text{and} \quad \bar{\sigma}[\tilde{y}] = \bigwedge_{i'\in\mathcal{Y}'} \gamma_{i'} \triangleleft \varepsilon^P_{i'} \tag{63}$$

Similarly for $\tilde{x}$ (that may have repeated names unlike $\tilde{y}$):

$$\sigma[\tilde{x}] = \bigwedge_{i\in\mathcal{X}} \gamma_i \triangleleft \varepsilon^P_i \quad \text{and} \quad \bar{\sigma}[\tilde{x}] = \bigwedge_{i'\in\mathcal{X}'} \gamma_{i'} \triangleleft \varepsilon^P_{i'} \tag{64}$$

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, activeness on a and $\bar{a}$ needs to be dropped. As we work with non-replicated prefixes, we can assume neither has ω 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 that activeness drops when applying the erasure operator. If both are plain then $\varepsilon' = \varepsilon = \bot$ so there's nothing to prove. If one is plain and the other is linear then only the plain one will be declared active, but then when composing Γ'_I and Γ'_O it is no longer observable, so, again, we get that neither a nor $\bar{a}$ is active in Γ'.

Starting from (59) we drop the activeness dependencies and replace $\sigma[\tilde{y}]$ by the corresponding resource set:

$$\Gamma'_I \succeq a : \sigma \odot a_{\mathbf{A}} \triangleleft \varepsilon \odot a_{\mathbf{R}} \triangleleft \left(\bar{l} \vee \bigwedge_{i \in \mathcal{Y}} \gamma_i \right) \odot \bar{\sigma}[\tilde{y}] \triangleleft (l \vee \bar{a}_{\mathbf{R}}) \odot \Gamma_I \tag{65}$$

Similarly to (61) we use assume the local component of (65) has the following normal form:

$$\bigwedge_{i \in \mathcal{Y}} \gamma_i \triangleleft \varepsilon'_i \ \wedge \ \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \varepsilon'_i \ \wedge \ a_{\mathbf{R}} \triangleleft \varepsilon_I. \tag{66}$$

The main difference between ε_i and ε'_i is due to dependencies getting reduced with $\bar{\sigma}[\tilde{y}]$. Their normal forms is similarly annotated with a tick $'$:

$$\forall i \in \mathcal{O} \cup \mathcal{Y} : \varepsilon'_i = \bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^{O} \wedge {\varepsilon'_{ij}}^{Y} \tag{67}$$

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 ε_I, by reducing $a_{\mathbf{R}} \triangleleft (\bar{l} \vee \bigwedge_{i \in \mathcal{Y}} \gamma_i)$ from (65) with statements in (67), dropping $\tilde{y}$-based dependencies and any other $a_{\mathbf{R}}$-dependency provided by Γ_I:

$$\varepsilon_I \succeq \bar{l} \vee \bigwedge_{i \in \mathcal{Y}} \bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^{O} \tag{68}$$

Combining (66) and (67) we can compute the behavioural statement in $(\boldsymbol{\nu} \tilde{y})\, \Gamma'_I$:

$$(a_{\mathbf{R}} \triangleleft \varepsilon_I) \wedge \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \left(\bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^{O} \wedge {\varepsilon'_{ij}}^{*} \right) \tag{69}$$

where ${\varepsilon'_{ij}}^{*}$ is one of $\bot$ (if $p_{\mathbf{A}} \succeq {\varepsilon'_{ij}}^{Y}$ for some p), $l \vee \bar{a}_{\mathbf{R}}$ (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 Ξ_{L} based on (60) and (69):

$$\Xi_{\mathrm{L}} = \bar{a}_{\mathbf{A}} \triangleleft \varepsilon' \odot \bar{a}_{\mathbf{R}} \triangleleft (\bar{l}' \vee \bar{\sigma}[\tilde{x}]) \odot \sigma[\tilde{x}] \triangleleft (l' \vee a_{\mathbf{AR}}) \odot \Gamma_O \triangleleft (l' \vee a_{\mathbf{A}}) \odot a_{\mathbf{R}} \triangleleft \varepsilon_I \wedge \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \bigvee_{j \in \mathcal{I}'_i} \left({\varepsilon'_{ij}}^{O} \wedge {\varepsilon'_{ij}}^{*} \right) \tag{70}$$

Dropping dependencies on $a_{\mathbf{A}}$, replacing $\bar{a}_{\mathbf{R}}$'s dependencies by the actual resource set, replacing $a_{\mathbf{R}}$'s dependencies using (68), and developing $\sigma[\tilde{x}]$ with (63) 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 ε_I, the strong dependency replaced by a weak one, and then the $a_{\mathbf{R}}$-term can be dropped, further strenghtening 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 $\bot$ they can (by the $\forall\varepsilon:\bot\succeq\varepsilon$ 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 $\bar{a}_{\mathbf{R}}{\triangleleft}(\bar{l}'\vee\bigwedge_{i\in\mathcal{X}'}\gamma_i)$-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

$$\bigwedge_{i\in\mathcal{X}}\gamma_i{\triangleleft}\left(\varepsilon_i^P\wedge\left(l'\vee\bar{l}\vee\bigvee_{j\in\mathcal{I}'_i}{\varepsilon'_{ij}}^O\right)\right)\odot\Gamma_O\odot$$
$$\bigwedge_{i\in\mathcal{O}}\gamma_i{\triangleleft}\left(\bigvee_{j\in\mathcal{I}'_i}{\varepsilon'_{ij}}^O\wedge\left(l\vee\bar{l}'\vee\left({\varepsilon'_{ij}}^Y\{\tilde{x}/\tilde{y}\}\right)\right)\right)\quad(71)$$

We now show that dropping the event annotations from that expression yields an equivalent type, building on the following lemma:

Lemma A.3.2 (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* $\bigwedge_{i,j} \left(\gamma_i \triangleleft (\varepsilon_i \wedge (\bar{l} \vee \varphi_i)) \wedge \gamma'_j \triangleleft ((l \vee \varepsilon'_j) \wedge \varphi'_j)\right) \cong \bigwedge_{i,j} \left(\gamma_i \triangleleft (\varepsilon_i \wedge \varphi_i) \wedge \gamma'_j \triangleleft (\varepsilon'_j \wedge \varphi'_j)\right)$.

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 ε'_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 (71) we will show that $\forall i' \in \mathcal{X}, i \in \mathcal{O}, j \in \mathcal{I}'_i$, either of the following hold

$$\varepsilon^P_{i'} \succeq {\varepsilon'_{ij}}^Y \tag{72}$$

$$\bigvee_{j' \in \mathcal{I}'_{i'}} {\varepsilon'_{i'j'}}^O \preceq {\varepsilon'_{ij}}^O \tag{73}$$

satisfying the conditions of the Lemma. Specifically, assume that (72) does *not* hold. As neither dependency in the inequality use disjunctions, there is α such that (for $\alpha' = \alpha\{^{\tilde{x}}/_{\tilde{y}}\}$) $\alpha' \preceq {\varepsilon'_{ij}}^X$,

$$\alpha \preceq {\varepsilon'_{ij}}^Y, \tag{74}$$

$$\alpha \not\preceq \varepsilon^P_{i'}. \tag{75}$$

Let $k \in \mathcal{Y}$ be such that $\gamma_k = \alpha$ (see (63)). By the definition of parameter instantiation (if $\gamma_{i'}$ does not depend on α then α depends on $\gamma_{i'}$), (75) implies

$$\gamma_{i'} \preceq \varepsilon^P_k. \tag{76}$$

As ${\varepsilon'_{ij}}^Y$ is taken from Γ'_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 (67) will give a type equivalent to Γ'_I.)

Inequality (74) can also be written ${\varepsilon'_{ij}}^Y \cong \alpha \wedge {\varepsilon'_{ij}}^Y$. Composing with the $\bar{\sigma}[\tilde{y}] \triangleleft (l \vee \bar{a}_{\mathbf{R}})$ term from (65), or more specifically $\gamma_k \triangleleft (l \vee (\bar{a}_{\mathbf{R}} \wedge \varepsilon^P_k))$ (remember that $\gamma_k = \alpha$), it becomes $(\alpha * (l \vee \bar{a}_{\mathbf{R}}) \wedge \varepsilon^P_k) \wedge {\varepsilon'_{ij}}^Y$. Applying (76) rewritten as $\varepsilon^P_k \cong \varepsilon^P_k \wedge \gamma_{i'}$ we get $(\alpha * (l \vee \bar{a}_{\mathbf{R}}) \wedge \varepsilon^P_k \wedge \gamma_{i'}) \wedge {\varepsilon'_{ij}}^Y$. As $i' \in \mathcal{O}$ we can apply (66) and get $(\alpha * (l \vee \bar{a}_{\mathbf{R}}) \wedge \varepsilon^P_k \wedge (\gamma_{i'} * \varepsilon'_{i'})) \wedge {\varepsilon'_{ij}}^Y$ where the meaning of the second $*$ depends on what kind of resource $\gamma_{i'}$ is. Rewriting $\varepsilon'_{i'}$ with (67) we get

$$\left(\alpha * (l \vee \bar{a}_{\mathbf{R}}) \wedge \varepsilon^P_k \wedge (\gamma_{i'} * \bigvee_{j' \in \mathcal{I}'_{i'}} {\varepsilon'_{i'j'}}^O \wedge {\varepsilon'_{i'j'}}^Y)\right) \wedge {\varepsilon'_{ij}}^Y \tag{77}$$

To summarise, ${\varepsilon'_{ij}}^Y$ can be replaced by (77) in Γ'_I (67), and the resulting type is equivalent, so it can be used instead of Γ'_I when computing (71).

Dependency (77) is strengthened by dropping $l \vee \bar{a}_{\mathbf{R}}$, ε^P_k and all ${\varepsilon'_{i'j'}}^Y$, and the two $*$ operators are handled like this: if they are conjunctions (for

responsiveness 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 $\tilde{y}$ (69), (77) becomes $\bigvee_{j' \in \mathcal{I}'_{i'}} {\varepsilon'_{i'j'}}^{O} \wedge {\varepsilon'_{ij}}^{*}$. Written in normal form (69), 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 (73).

We can therefore apply Lemma A.3.2 to (71) twice (for l and then l'), getting

$$\Gamma \succeq \bigwedge_{i \in \mathcal{X}} \gamma_i \triangleleft \left(\bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^{O} \wedge \varepsilon_i^P \right) \odot \Gamma_O \odot \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \left(\bigvee_{j \in \mathcal{I}'_i} {\varepsilon'_{ij}}^{O} \wedge \left({\varepsilon'_{ij}}^{Y} \{ {}^{\tilde{x}}/_{\tilde{y}} \} \right) \right) \tag{78}$$

The factors ε_i^P can now be strengthened to ${\varepsilon'_{ij}}^{Y}\{{}^{\tilde{x}}/_{\tilde{y}}\}$ and, comparing with (67), observe that the dependencies of γ_i for $i \in \mathcal{X}$ and $\mathcal{O}$ are exactly $\varepsilon'_i\{{}^{\tilde{x}}/_{\tilde{y}}\}$:

$$\Gamma \succeq \bigwedge_{i \in \mathcal{X}} (\gamma_i \triangleleft \varepsilon'_i)\{{}^{\tilde{x}}/_{\tilde{y}}\} \odot \Gamma_O \odot \bigwedge_{i \in \mathcal{O}} \gamma_i \triangleleft \left(\varepsilon'_i \{ {}^{\tilde{x}}/_{\tilde{y}} \} \right) \tag{79}$$

As substitution distributes on composition we get $\Gamma \succeq \Gamma'_I\{{}^{\tilde{x}}/_{\tilde{y}}\} \odot \Gamma_O$. In order to reach Γ' we still need to transform Γ'_I into Γ_I, i.e. cancel the composition of Γ_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 ε_i. Applying the parameter instantiation (63) to it replaces it with $\gamma_{i'} * (\bar{a}_{\mathbf{R}} \wedge \varepsilon_{i'}^P)$. If $\gamma_{i'}$ is a responsiveness resource, this can be immediately strengthened back to $\gamma_{i'}$. If it is an activeness resource, then the $\bar{a}_{\mathbf{R}} \triangleleft \bar{\sigma}[\tilde{x}]$ term from Γ'_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.3.2 Output

Let $\Gamma \vdash P \xrightarrow{\overline{a}\langle \tilde{x} \rangle} P'$. 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 (R-Sum), and if a branching consumed by the transition is active in Γ ($(\sum_i p_i)_{\mathbf{A}}$), then it is removed as specified by Definition 6.3.7. We can now proceed to the sum-less case.

Consider the transition $P = \overline{a}\langle \tilde{x} \rangle.Q \xrightarrow{\overline{a}\langle \tilde{x} \rangle} Q$.

Assuming $\Gamma \vdash Q$, we get the following type for P (with $\varepsilon \in \{\top, \bot\}$ depending on a's multiplicities):

$$\Gamma' = a : \sigma \odot \bar{a}_{\mathbf{A}} \triangleleft \varepsilon \odot \bar{a}_{\mathbf{R}} \triangleleft (\bar{l} \vee \bar{\sigma}[\tilde{x}]) \odot \sigma[\tilde{x}] \triangleleft (l \vee a_{\mathbf{AR}}) \odot \Gamma \triangleleft (l \vee a_{\mathbf{A}}) \tag{80}$$

Having $\Gamma'' = \Gamma' \wr \overline{a}\langle \tilde{x} \rangle$, we want to show that $\Gamma \preceq \Gamma''$.

Recall that

$$\Gamma'' \stackrel{\text{def}}{=} \Gamma' \wr \bar{a} \otimes \bar{\sigma}[\tilde{x}] \triangleleft (\bar{a}_{\mathbf{R}} \blacktriangleleft a_{\mathbf{R}}) \tag{81}$$

We first show that multiplicities in Γ'' are equal or weaker than the ones in Γ, before proceeding to the behavioural statement.

Simplifying (80) and (81) 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,

$$\#\Gamma'' \succeq (\bar{a} \odot \#\Gamma) \wr \bar{a} \tag{82}$$

Let a's multiplicities in Γ 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

$$(\bar{a} \odot \Gamma) \wr \bar{a} \succeq \Gamma \tag{83}$$

Note that $\wr\bar{a}$ and $\backslash\bar{a}$ coincide in this case, as $m_o + 1 \neq \omega$.

Composing (82) and (83) gives us $\#\Gamma'' \succeq \#\Gamma$, and we now proceed to the behavioural statement.

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 $\Omega = |\bar{\sigma}[\tilde{x}]| \setminus \{\bar{a}_{\mathbf{A}}, \bar{a}_{\mathbf{R}}\}$, the set of resources to be provided by the output, and $T = (\Omega \cup |\sigma[\tilde{x}]|) \setminus \{\bar{a}_{\mathbf{A}}, \bar{a}_{\mathbf{R}}\}$ the set of resources to be provided on one side or the other. In both cases we exclude $\bar{a}_{\mathbf{AR}}$ because they interact with the statements introduced by the (R-Pre) rule and have to be handled specially.

We show that each dependency statement in Γ'' is also present in Γ, in a possibly weaker form.

Dependencies in Γ'' are partitioned as follows:

1. $\{\bar{a}_{\mathbf{A}}, \bar{a}_{\mathbf{R}}\}$
2. Ω
3. $T \setminus \Omega$
4. $|\Gamma| \setminus (\{\bar{a}_{\mathbf{A}}, \bar{a}_{\mathbf{R}}\} \cup T)$

We cover each of those classes in order.

1. $\{\bar{a}_{\mathbf{A}}, \bar{a}_{\mathbf{R}}\}$

Output $\bar{a}$-activeness in Γ' and Γ'' may be provided by four different terms. In the following a missing $\bar{a}$-activeness statement is written $\bar{a}_{\mathbf{A}} \triangleleft \bot$.

- $\bar{a}_{\mathbf{A}} \triangleleft \varepsilon$ as given by the (R-Pre) rule,
- $\bar{a}_{\mathbf{A}} \triangleleft \varepsilon_c \in \Gamma$,
- $\bar{a}_{\mathbf{A}} \triangleleft \varepsilon_i \in \sigma[\tilde{x}]$,
- $\bar{a}_{\mathbf{A}} \triangleleft \varepsilon_o \in \bar{\sigma}[\tilde{x}]$.

In (81), the $\Gamma' \wr \bar{a}$ type contains $\bar{a}_{\mathbf{A}} \triangleleft \bot$, by definition of that operator. The $\otimes\overline{\sigma}[\tilde{x}]$-operation preserves that statement (as it may only weaken activeness statements), so $\bar{a}_{\mathbf{A}}$'s dependencies in Γ'' are equal to those in $\bar{\sigma}[\tilde{x}]$ (after reducing the $\bar{a}_{\mathbf{R}}$-dependency), i.e. $\bar{a}_{\mathbf{A}} \triangleleft (\varepsilon' \wedge \varepsilon_o) \in \Gamma''$ where ε' is $\bar{a}_{\mathbf{R}}$'s dependencies in Γ''.

First assume $\varepsilon_o \cong \bot$. Then $\bar{a}_{\mathbf{A}} \triangleleft \bot \in \Gamma''$. 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 $\bar{a}_{\mathbf{A}} \in |\bar{\sigma}[\tilde{x}]|$, i.e. $\bar{a}_{\mathbf{R}} \triangleleft \bar{a}_{\mathbf{A}} \preceq \Gamma'$, which reduces with $\bar{a}_{\mathbf{A}} \triangleleft \bot$ to give $\bar{a}_{\mathbf{R}} \triangleleft \bot$, i.e. $\varepsilon' \cong \bot$, which itself causes $\bar{a}_{\mathbf{A}} \triangleleft \bot \in \Gamma''$, 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 $\bar{a}_{\mathbf{A}} \in |\bar{\sigma}[\tilde{x}]|$, Convention 6.2.4 applies to forbid the type to have blocked activeness, i.e. there is $\bar{a}^m \in \sigma[\tilde{x}]$ with $m > 0$. Because (R-Pre) introduces that type into Γ', $m_i > 0$ as well. The sum of two non-zero multiplicities being $\star$,[9] the side condition in (R-Pre) requires $m_o + m'_o \notin \{1; \star\}$. Since Γ' includes $\bar{a}^1_{\mathbf{A}} \triangleleft \varepsilon$, $m_o > 1$. 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.

For responsiveness, let $\bar{a}_{\mathbf{R}} \triangleleft \varepsilon_0 \in \Gamma$. Then $\exists \varepsilon' : \bar{a}_{\mathbf{R}} \triangleleft \varepsilon' \in \Gamma'$ and $\varepsilon' \succeq \varepsilon_0$ (the exact value is given in (85) below). Then let $\bar{a}_{\mathbf{R}} \triangleleft \varepsilon'' \in \Gamma''$. We have $\varepsilon'' \succeq \varepsilon'$ so that $\varepsilon'' \succeq \varepsilon_0$, as required.

2. Ω

We first calculate $\bar{a}_{\mathbf{R}}$'s dependencies in Γ'.

Having $\forall \alpha \in \Omega : \alpha \triangleleft \varepsilon_\alpha \in \Gamma$, fix a set of $\varepsilon_{\alpha i}$ and $\varepsilon'_{\alpha i}$ and indexing sets I_α such that:

$$\varepsilon_\alpha \cong \bigvee_{i \in I_\alpha} (\varepsilon_{\alpha i} \wedge \varepsilon'_{\alpha i}) \tag{84}$$

and $|\varepsilon_{\alpha i}| \cap \Omega = \varnothing$, $|\varepsilon'_{\alpha i}| \subseteq \Omega$ for all α and i.

Let Φ be the set of functions φ 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 ε_φ 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:

$$\varepsilon' \cong \varepsilon_0 \wedge \bigvee_{\varphi \in \Phi} \left(\varepsilon_\varphi \wedge \bigwedge_{\alpha \in \Omega} \varepsilon'_{\alpha\varphi(\alpha)} \right) \tag{85}$$

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

$$\varepsilon''_\alpha \succeq \bigvee_{\varphi \in \Phi} \left(\varepsilon_{\alpha 0} \ \wedge \ \varepsilon_\varphi \ \wedge \bigwedge_{\alpha' \in \Omega} \varepsilon'_{\alpha'\varphi(\alpha')} \right) \tag{86}$$

That equation gives a stronger form of ε''_α where we removed ε_0 from (85) as well as statements for resources α contained in $|\sigma[\tilde{x}]| \cap |\bar{\sigma}[\tilde{x}]|$, produced by $\sigma[\tilde{x}]$ in Γ'. Note that those statements may only be responsiveness statements

[9] 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 $\overline{1}_{\mathbf{A}}$ in ξ_O and 1^{m_2} in ξ_I would not have blocked activeness but subject reduction would not hold for transitions like $\overline{a}\langle a \rangle \xrightarrow{\overline{a}\langle a \rangle} \mathbf{0}$.

by Convention 6.2.4, and will therefore be added to ε_α 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 Γ are contained in the protocol (to be precise, by parameter instantiation of local resources, which includes dependencies added to complete the protocol).

Lemma A.3.3 (Protocol Satisfaction) *Let Ω, Φ, $\varepsilon_{\alpha i}$ 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 α, $\varepsilon_{\alpha\varphi(\alpha)} \preceq \varepsilon_{\alpha 0}$.

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

$$\tilde{\beta}_s \subseteq \tilde{\beta}'_s \quad \wedge \quad \tilde{\beta}_w \subseteq \tilde{\beta}'_w. \tag{87}$$

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 φ 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 (87) 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 (86) is weaker than $\alpha \triangleleft \varepsilon_\alpha$ which is in Γ:

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 φ is at a circularity then the inequality is an immediate consequence of $\forall\varepsilon : \varepsilon \preceq \bot$. if φ is not at a circularity then $\varepsilon_\varphi = \top$ and the inequality is proved in Lemma A.3.3.

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. $T \setminus (\bar{a}_{\mathbf{AR}} \cup \Omega)$

Let $\alpha \in T$ (and not in $\bar{a}_{\mathbf{AR}} \cup \Omega$), with $\alpha \triangleleft \varepsilon_\alpha \in \Gamma$, $\varepsilon'_\alpha \in \sigma[\tilde{x}]$. Then $\varepsilon_\alpha \wedge \varepsilon'_\alpha \in \Gamma'$, with an additional $a_{\mathbf{AR}}$-dependency if $l \notin \tilde{l}$. Then, by definition of the "$\otimes(\overline{\sigma}[\tilde{x}] \triangleleft (l \vee \bar{a}_{\mathbf{AR}}))$" operation, $\exists\alpha'' \triangleleft \varepsilon$ s.t. $\varepsilon''_\alpha \in \Gamma''$ and $\varepsilon''_\alpha \preceq \varepsilon_\alpha$.

4. $|\Gamma| \setminus (\{\bar{a}_{\mathbf{A}}, \bar{a}_{\mathbf{R}}\} \cup T)$

Those resources have, in both Γ' and Γ'' and compared to Γ, just one $a_{\mathbf{A}}$ additional dependency which can be removed by strengthening.

A.3.3 Input

Let $(\Gamma'; P') \xrightarrow{a(\tilde{x})} (\Gamma''; P'')$, where $P' = a(\tilde{y}).P$ and $\Gamma' \vdash P'$.

Then the type system types $P_0 = P' \,|\, \overline{a}\langle\tilde{x}\rangle$ with a type Γ_0 equal to Γ'' but with an additional $a_{\mathbf{A}}$-dependency on $\overline{\sigma}[\tilde{x}]$, that can be removed by strengthening (i.e. $\Gamma_0 \preceq \Gamma''$). As $P_0 \xrightarrow{\tau} P''$, Γ'' we can apply the τ-case of subject reduction (proved in Section A.3.1), and have $\Gamma'_0 \vdash P''$ for some Γ'_0 with $\Gamma'_0 \preceq \Gamma_0$.

A.3.4 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).

Following the type system rule (R-Pre), P's type Γ is as follows:

$$\Gamma = \left(p : \sigma; p^{\omega}_{\mathbf{A}} \blacktriangleleft p^{m} \wedge \bar{p}^{m'}\right) \odot\, !\, (\boldsymbol{\nu}\tilde{z})\, \left(p_{\mathbf{R}} \triangleleft \sigma[\tilde{y}] \odot \overline{\sigma}[\tilde{y}] \triangleleft \bar{p}_{\mathbf{AR}} \odot \Gamma_0 \triangleleft \bar{p}_{\mathbf{A}}\right) \odot \Gamma_Q \quad (88)$$

where $\Gamma_0 \vdash P_0$ and $\Gamma_Q \vdash Q$.

The proof involves extracting one element of the replicated process (as if we invoked the usual rule $!\, P \mapsto (P \mid !\, P)$, which, remember, is not part of out notion of structural congruence because a port with multiplicity ω 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 (88), $\hat{P}$ has type $\hat{\Gamma}$:

$$\begin{aligned}\hat{\Gamma} = {} & \left(q : \sigma; q^{1} \blacktriangleleft\right) \odot (\boldsymbol{\nu}\tilde{z})\, \left(q_{\mathbf{R}} \triangleleft \sigma[\tilde{y}] \odot \overline{\sigma}[\tilde{y}] \triangleleft \bar{q}_{\mathbf{AR}} \odot \Gamma_0 \triangleleft \bar{q}_{\mathbf{A}}\right) \odot \\ & \left(p : \sigma; p^{\omega}_{\mathbf{A}} \blacktriangleleft p^{m} \wedge \bar{p}^{m'-1}\right) \odot\, !\, (\boldsymbol{\nu}\tilde{z})\, \left(p_{\mathbf{R}} \triangleleft \sigma[\tilde{y}] \odot \overline{\sigma}[\tilde{y}] \triangleleft \bar{p}_{\mathbf{AR}} \odot \Gamma_0 \triangleleft \bar{p}_{\mathbf{A}}\right) \odot \Gamma_Q\end{aligned} \quad (89)$$

Observe that we set $\bar{p}$'s remote multiplicities to $m' - 1$ rather than just m' like in (88), 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 $\hat{\Gamma} \vdash \hat{P}$ as (R-Pre) doesn't put any restriction on remote multiplicities of the complement.

Define a set of Γ_i and $\hat{\Gamma}_i$ such that:

$$\overline{\sigma}[\tilde{y}] \triangleleft \bar{p}_{\mathbf{AR}} \odot \Gamma_0 \triangleleft \bar{p}_{\mathbf{A}} = \bigvee_{i \in I} \Gamma_i \quad (90)$$

$$\overline{\sigma}[\tilde{y}] \triangleleft \bar{q}_{\mathbf{AR}} \odot \Gamma_0 \triangleleft \bar{q}_{\mathbf{A}} = \bigvee_{i \in I} \hat{\Gamma}_i \quad (91)$$

As q is fresh, $\mathrm{n}(q)$ doesn't appear in Γ_0 or $\overline{\sigma}[\tilde{y}]$ and

$$\forall i \in I : \hat{\Gamma}_i\{^{\mathrm{n}(p)}/_{\mathrm{n}(q)}\} = \Gamma_i \quad (92)$$

As $\odot$ and $\boldsymbol{\nu}\tilde{z}$ are logical homomorphisms, the q-part from (89) under $\tilde{z}$-

replication is:

$$\begin{aligned}(\boldsymbol{\nu}\tilde{z})\left(q_{\mathbf{R}}\triangleleft\sigma[\tilde{y}]\odot\overline{\sigma}[\tilde{y}]\triangleleft\bar{q}_{\mathbf{AR}}\odot\Gamma_0\triangleleft\bar{q}_{\mathbf{A}}\right) &= (\boldsymbol{\nu}\tilde{z})\bigvee_{i\in I}\left(q_{\mathbf{R}}\triangleleft\sigma[\tilde{y}]\odot\hat{\Gamma}_i\right)\\ &= \bigvee_{i\in I}(\boldsymbol{\nu}\tilde{z})\left(q_{\mathbf{R}}\triangleleft\sigma[\tilde{y}]\odot\hat{\Gamma}_i\right)\\ &\cong \bigvee_{i\in I}\left(q_{\mathbf{R}}\triangleleft\hat{\varepsilon}_i\wedge(\boldsymbol{\nu}\tilde{z})\,\hat{\Gamma}_i\right)\end{aligned}\tag{93}$$

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,

$$(\boldsymbol{\nu}\tilde{z})\left(p_{\mathbf{R}}\triangleleft\sigma[\tilde{y}]\odot\overline{\sigma}[\tilde{y}]\triangleleft\bar{p}_{\mathbf{AR}}\odot\Gamma_0\triangleleft\bar{p}_{\mathbf{A}}\right)\cong\bigvee_{i\in I}\left(p_{\mathbf{R}}\triangleleft\varepsilon_i\wedge(\boldsymbol{\nu}\tilde{z})\,\Gamma_i\right)$$

for some set of ε_i. Replicating that type gives:

$$\mathbf{!}\,(\boldsymbol{\nu}\tilde{z})\left(p_{\mathbf{R}}\triangleleft\sigma[\tilde{y}]\odot\overline{\sigma}[\tilde{y}]\triangleleft\bar{p}_{\mathbf{AR}}\odot\Gamma_0\triangleleft\bar{p}_{\mathbf{A}}\right)\cong\bigvee_{J\subseteq I}\bigodot_{j\in J}\left(p_{\mathbf{R}}\triangleleft\varepsilon_j\odot(\boldsymbol{\nu}\tilde{z})\,\Gamma_j^2\right)\tag{94}$$

We are now ready to compute $\Gamma\wr\mu$ and $\hat{\Gamma}\wr\hat{\mu}$. Since the definition of $\wr$ (Definition 6.3.12, page 37) 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 μ 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 (94):

$$\Gamma\wr\mu\succeq\left(\left(p:\sigma;p_{\mathbf{A}}^{\omega}\blacktriangleleft p^m\wedge\bar{p}^{m'-1}\right)\odot\bigvee_{J\subseteq I}\bigodot_{j\in J}\left(p_{\mathbf{R}}\triangleleft\varepsilon_j\wedge(\boldsymbol{\nu}\tilde{z})\,\Gamma_j^2\right)\right)\otimes\overline{\sigma}[\tilde{x}]\triangleleft p_{\mathbf{R}}\tag{95}$$

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 (95), we may move $p_{\mathbf{R}}\triangleleft\varepsilon_j$ outside the composition:

$$\Gamma\wr\mu\succeq\left(\left(p:\sigma;p_{\mathbf{A}}^{\omega}\blacktriangleleft p^m\wedge\bar{p}^{m'-1}\right)\odot\bigvee_{i\in I}\left(p_{\mathbf{R}}\triangleleft\varepsilon_i\wedge\bigvee_{\substack{J\subseteq I\\ J\ni i}}\bigodot_{j\in J}(\boldsymbol{\nu}\tilde{z})\,\Gamma_j^2\right)\right)\otimes\overline{\sigma}[\tilde{x}]\triangleleft p_{\mathbf{R}}\tag{96}$$

Moving on to $\hat{\Gamma}$ and $\hat{\Gamma}\wr\hat{\mu}$:

$$\begin{aligned}\hat{\Gamma}\wr\hat{\mu}\preceq\Bigg(&\left(p:\sigma;p_{\mathbf{A}}^{\omega}\blacktriangleleft p^m\wedge\bar{p}^{m'-1}\right)\odot\bigvee_{J\subseteq I}\bigodot_{j\in J}\left(p_{\mathbf{R}}\triangleleft\varepsilon_j\wedge(\boldsymbol{\nu}\tilde{z})\,\Gamma_j^2\right)\\ &\odot\left(q:\sigma;q^0\blacktriangleleft\ \right)\odot\bigvee_{i\in I}\left(q_{\mathbf{R}}\triangleleft\hat{\varepsilon}_i\wedge(\boldsymbol{\nu}\tilde{z})\,\hat{\Gamma}_i\right)\Bigg)\otimes\overline{\sigma}[\tilde{x}]\triangleleft q_{\mathbf{R}}\end{aligned}\tag{97}$$

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 Γ_0 (the continuation). As $\triangleleft$ is contravariant on the right wrt. $\preceq$ (Definition 6.3.1, page 33), $\forall i : q_{\mathbf{R}} \triangleleft \hat{\varepsilon}_i \preceq q_{\mathbf{R}} \triangleleft \varepsilon_i$, so (97) becomes

$$\hat{\Gamma} \wr \hat{\mu} \preceq \left(\left(p : \sigma; p^{\omega}_{\mathbf{A}} \blacktriangleleft p^m \wedge \bar{p}^{m'-1} \right) \odot \bigvee_{J \subseteq I} \bigodot_{j \in J} \left(p_{\mathbf{R}} \triangleleft \varepsilon_j \wedge (\boldsymbol{\nu} \tilde{z})\, \Gamma^2_j \right) \right.$$
$$\left. \odot \bigvee_{i \in I} \left(q_{\mathbf{R}} \triangleleft \varepsilon_i \wedge (\boldsymbol{\nu} \tilde{z})\, \hat{\Gamma}_i \right) \right) \otimes \overline{\sigma}[\tilde{x}] \triangleleft q_{\mathbf{R}} \quad (98)$$

Let's call that type Γ_M ("M" as it is in some sense "in the Middle" between $\Gamma \wr \mu$ and $\hat{\Gamma} \wr \hat{\mu}$). Inequality (98) can then be written $\Gamma_M \succeq \hat{\Gamma} \wr \hat{\mu}$, or

$$\Gamma_M \{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \succeq \hat{\Gamma} \wr \hat{\mu} \{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \quad (99)$$

Applying (92) and the definition of replication ($\Gamma \odot \mathbf{!}\,\Gamma = \mathbf{!}\,\Gamma$), (96) becomes

$$\Gamma \wr \mu \succeq \Gamma_M \{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \quad (100)$$

We have already shown subject reduction for transitions using non-replicated guards, so there is Γ' such that $\hat{\Gamma}' \wr \hat{\mu} \succeq \Gamma'$ and $\Gamma' \vdash P'$. The first equation implies

$$\hat{\Gamma}' \wr \hat{\mu} \{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \succeq \Gamma' \{^{\mathrm{n}(q)}/_{\mathrm{n}(p)}\} \quad (101)$$

As $\mathrm{n}(q)$ doesn't appear in P', it doesn't appear in Γ' either so

$$\Gamma' \{^{\mathrm{n}(p)}/_{\mathrm{n}(q)}\} = \Gamma' \quad (102)$$

Chaining (100), (99), (101) and (102) we get $\Gamma \wr \mu \succeq \Gamma'$, as required.

A.4 Simple Correctness

As a pre-requisite to soundness we show the following lemma:

Lemma A.4.1 (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.4.2 *Let $\Gamma = (\Sigma; \Xi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ and Γ' 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 ω-names:

Lemma A.4.3 *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 ω for prefix subjects
2. Prefix rules $\odot$-compose the remote behaviour (for objects)

Therefore p^{ω} 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 ω 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^{ω} 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 $\mathrm{n}(p)$). □

We work on a restricted form of simple correctness that does not permit arbitrary transition sequences:

Definition A.4.4 (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 4.4.1 whenever* $\tilde{\mu} = \varnothing$.

Then this lemma, together with Subject Reduction, will be used for full generality:

Lemma A.4.5 (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.4.6 (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 4.4.1.

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

$$\Gamma = (\boldsymbol{\nu}\tilde{z})\,((\boldsymbol{\nu}\mathrm{bn}(G))\,(\Big(p : \sigma;\ \blacktriangleleft\, p^{m_0} \wedge \bar{p}^{m'_0}\Big) \odot p_{\mathbf{A}}^{\#(G)} \triangleleft \varepsilon \odot p_{\mathbf{R}} \triangleleft (\bar{l} \vee \sigma[\tilde{x}]) \odot \bar{\sigma}[\tilde{x}] \triangleleft (l \vee \bar{p}_{\mathbf{AR}}) \odot \Gamma_Q \triangleleft (l \vee \bar{p}_{\mathbf{A}}))) \odot \Gamma_R \quad (103)$$

In (103), m_0 must be equal to $\#(G)$ or $\star$ in order for the first composition to be well-defined, so, by Lemma A.4.2, $p^m \succeq \Xi_{\mathrm{L}}$ 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 μ' equal to μ 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.4.3, 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 $P \equiv (\boldsymbol{\nu}\tilde{z})\,(!\,(\boldsymbol{\nu}\tilde{z}')\,Q \,|\, R)$ with $R = a(\tilde{y}).R'$ (for $p = a$) or $R = \overline{a}\langle\tilde{x}\rangle.R'$ (for $p = \overline{a}$). The $\exists! Q$ s.t. $P \xrightarrow{\mu} Q$ condition is then immediately satisfied as μ must use that prefix, with the objects given by μ.

□

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 Γ'' such that $\Gamma'' \vdash P'$ and $\Gamma'' \succeq \Gamma'$. By the Local Correctness Lemma, $\mathsf{good}_{\#}(\Gamma''; P')$. By Lemma A.4.5, $\mathsf{good}_{\#}(\Gamma'; P')$ as well. Since this is valid for any transition sequence $\tilde{\mu}$, $\Gamma \models_{\#} P$.

A.5 Soundness

In this section we prove the soundness lemma, i.e. that $\Gamma \vdash P$ implies $\Gamma \models P$.

In the previous section we showed that $(\Gamma; P)$ is correct with respect to simple semantics. We now proceed with the rest of the semantic definition.

Before proceeding to the proof itself we construct a framework that lets us unambiguously address individual channels, taking into account that in processes such as $!\,a.(\boldsymbol{\nu}t)\,P$ there are potentially infinitely many distinct instances of t, as the process is replicated. Secondly, that framework permits addressing individual occurrences of channel names throughout the evolution of a process. Thirdly we introduce a compact notation (called *activeness strategy*) for transition sequences expressed up to certain permutation/deletion of unrelated transitions.

The proof is then merely a matter of constructing such activeness strategies in the typing rules and verifying they satisfy certain consistency criteria.

Specifically (the numbers in bracket are Definition numbers):

1. *Extended process syntax* (Section A.5.1): The process syntax is extended to incorporate events l (see Section 7.1), so that the connection between events in dependencies and the corresponding prefix in the process is maintained. This will also permit giving a precise semantics to dependencies such as $\bar{l} \vee \varepsilon$ and $l \vee \varepsilon$. An *annotated form* (A.5.2) P' of a process P is one that uses that extended syntax, and removing the annotation is done using the operator $\mathrm{ran}(P') = P$ (A.5.1).

2. *Strategies in Process Types* (Section A.5.2): In order to keep track of how dependency statements were obtained we annotate dependency statements $\gamma \triangleleft \varepsilon$ with *activeness strategies* ρ (A.5.4) and *responsiveness strategies* ϕ (A.5.8) that indicates which events must be triggered (and how) for a resource to become available. *Annotated activeness statements* are then of the form $p_{\mathbf{A}} \triangleleft \varepsilon \colon \rho$ and annotated *responsiveness* statements are of the form $p_{\mathbf{R}} \triangleleft \varepsilon \colon \rho.\, \phi$ where ρ and ϕ respectively describe a p-prefix and the corresponding continuation.

   A strategy ρ is essentially a binary tree whose nodes are individual events, and siblings are communication partners. This makes it easy to gradually modify a strategy when components of a process are pieced together and when dependencies are reduced. Is is also easy to translate an activeness strategy into an actual transition sequence. Again, removing the strategy annotations is done by the $\mathrm{ran}(\Gamma')$ operator (A.5.11).

3. *Consistency Criteria* (Section A.5.3): We propose *consistency criteria* (A.5.13 and A.5.16) on annotated process types to guarantee that activeness strategies can actually (*in the absence of interference*) be translated to sequences of transitions in the process, and that the resources required by those sequences (A.5.14 and A.5.15) do not exceed the dependencies declared in the process type.

4. *Interference and Completeness* (Section A.5.4)

   In a particular run a process makes a number of *choices*, for instance by selecting a particular communication partner for a non-replicated guard. We represent individual choices with activeness strategies, called *selection strategies*. Noting that two choices can be incompatible or *contradictory* (A.5.19) if they can't both be made in a single process run, a set of choices $\tilde{\rho}_c$ made in a particular run of the process is called a *choice set* (A.5.20).

   Similarly, an activeness strategy is either compatible or *contradicts* a given choice set. For a typing $\Gamma \models P$ to be semantically correct it needs to be prepared to face arbitrary interference (in the form of $\tilde{\mu}_i$-transitions with $i \notin I$ in Definition 6.4.4), in other words Γ must be *complete* (A.5.21): $\Gamma \cong \bigvee_{j \in J} \Gamma_j$ where for any possible choice set $\tilde{\rho}_c$ there is $j \in J$ such that Γ_j does not use any strategy contradicting $\tilde{\rho}_c$.

5. *Annotated transition system* (Section A.5.5): In order to prove that consistency and completeness form a sound characterisation of $\Gamma \models P$, the labelled transition system is extended (A.5.24) to update annotations in the process as well as strategies in the type while preserving consistency and completeness.

6. *Annotated type system* (Section A.5.6): We augment the type system from Section 6.5, so $\Gamma \vdash' P$ (A.5.42) extracts event names from an annotated process rather than picking new ones, and constructs annotated behavioural statements.

The soundness proof is based on a number of lemmas highlighting properties of the above setting:

1. *Type system equivalence* (A.5.43): If a $(\Gamma; P)$ is accepted by the original type system then there is an annotated typed process $(\Gamma'; P')$ such that $\Gamma' \vdash' P'$ and $\mathrm{ran}(\Gamma'; P') = (\Gamma; P)$.

2. *Soundness of the Annotated Type System* (A.5.44) says that the annotated type system produces types that are consistent and complete.

   Some intermediary types are not complete (the $\odot$-factors of the (R-Pre) and (R-Par) rules), but their closure is, motivating the *pre-completeness* property (Definition A.5.40), that is satisfied by those factors. Lemma A.5.41 shows that the closure of a pre-complete type is complete.

3. *Completeness and Correctness Lemma* (A.5.31) shows that if an annotated typed process $(\Gamma; P)$ is consistent and complete then $\Gamma \models P$ (restricted to the case where, in Definition 6.4.4, $\tilde{\mu}_q = \varnothing$). It builds on the following two lemmas:

4. *Strategy Application Lemma* (Lemma A.5.30) provides a definition to the strategy f in Definition 6.4.4. It also deals with the liveness requirement ("$\exists i \in I$" in Definition 6.4.4) by showing that the *application* of a strategy cause some form of *progress* towards reaching a resource, the progress being measured by decreasing *weight* of strategies.

5. *Runnability Safety Lemma* (Lemma A.5.29) deals with the robustness requirement (the $\{\mu_i\}_{i \notin I}$ in Definition 6.4.4) by showing that transitions preserve consistency and completeness of annotated process types.

A.5.1 Events in Processes

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 7.1 into the process syntax. Specifically, an *annotated process* is any production from P in the grammar below.

$$
\begin{array}{lcl}
P & ::= & (\boldsymbol{\nu}\mathfrak{x})\,P \;\mid\; P_{\mathsf{par}} \\
P_{\mathsf{par}} & ::= & (P_{\mathsf{par}} \,|\, P_{\mathsf{par}}) \;\mid\; P_{\mathsf{sum}} \;\mid\; \mathbf{0} \\
P_{\mathsf{sum}} & ::= & (P_{\mathsf{sum}} + P_{\mathsf{sum}}) \;\mid\; G^{\mathfrak{l}}.P \\
G & ::= & \mathbf{!}\,G_{\mathsf{norep}} \;\mid\; G_{\mathsf{norep}} \\
G_{\mathsf{norep}} & ::= & (\boldsymbol{\nu}\mathfrak{x})\,G_{\mathsf{norep}} \;\mid\; \overline{\mathfrak{a}}\langle\tilde{\mathfrak{x}}\rangle \;\mid\; \mathfrak{a}(\tilde{\mathfrak{y}}) \\
\text{Extended event: } \mathfrak{l} & ::= & \mathfrak{l}.\mathfrak{l} \;\mid\; \bullet.\mathfrak{l} \;\mid\; l \\
\text{Extended name: } \mathfrak{a},\mathfrak{x},\mathfrak{y} & ::= & \mathfrak{l}.\mathfrak{x} \;\mid\; \bullet.\mathfrak{l} \;\mid\; x
\end{array}
$$

Extended names are used to distinguish between different private channels with the same name. for instance, using the standard π-calculus transition rules (ignoring the $\mathfrak{l}$-annotations), a τ-transition on a in $\mathbf{!}\,a^{\mathfrak{l}}.(\boldsymbol{\nu}n)\,P \,|\, \bar{a}^{\mathfrak{l}'}$ would result in the process $\mathbf{!}\,a^{\mathfrak{l}}.(\boldsymbol{\nu}n)\,P \,|\, (\boldsymbol{\nu}n)\,P$, that has two distinct channels with the same name n. Using extended names we can write the resulting process $\mathbf{!}\,a^{\mathfrak{l}}.(\boldsymbol{\nu}n)\,P \,|\, (\boldsymbol{\nu}\mathfrak{l}'.\,n)\,P'$, 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 100 and following). Up to that point the reader

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 A.5.1 (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_{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 A.5.2 (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 A.5.3 *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 obtained by α-renaming bound names and for instance numbering all guards from left to right.

A.5.2 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 activeness dependency statement.

Formally:

Definition A.5.4 (Activeness Strategy) *Strategies are produced by the following grammar:*

$$
\begin{array}{rcl}
\rho & ::= & \tilde{\pi}\,\mathfrak{k}\,(\tilde{\pi})\delta \;\;\big|\;\; \pi\delta \;\;\big|\;\; \mathfrak{s} \\
\mathfrak{s} & ::= & \mathfrak{s}{+}\mathfrak{l} \;\;\big|\;\; \mathfrak{l} \\
\delta & ::= & .\,\rho \;\;\big|\;\; [\,s\,] \\
\pi & ::= & (\mathfrak{l}|\rho) \;\;\big|\;\; (\mathfrak{l}|\rho] \;\;\big|\;\; (\mathfrak{l}|\bullet) \;\;\big|\;\; (\rho|\bullet] \;\;\big|\;\; (\bullet|\rho) \\
\tilde{\pi} & ::= & \pi.\,\tilde{\pi} \;\;\big|\;\; \pi
\end{array}
$$

An annotated activeness dependency *is an expression of the form*

$$s_{\mathbf{A}} \lhd \varepsilon : \rho,$$

read "Strategy ρ provides an s-prefix and depends on ε".

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 A.5.5 (Top-Level and Guards) *A sum* $\mathfrak{s} = \sum_{i\in I} \mathfrak{l}_i$ *is* at 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 ρ_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 τ). 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 A.5.13). 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 ρ 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 ρ (note that the left bracket is still round, to emphasise the fact that $\mathfrak{l}$ must be accurately followed, unlike ρ). Consider the process

$$P = a(y)^{l_a}.(\bar{s}^{l_{\bar{s}}} \mid \bar{y}^{l_{\bar{y}}}.\bar{t}^{l_{\bar{t}}}) \mid \overline{a}\langle b\rangle^{l_1} \mid \overline{a}\langle c\rangle^{l_2} \mid b^{l_b}\ . \tag{104}$$

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 $\pi_1.\,\pi_2.\,\cdots.\,\pi_n\not\downarrow(\tilde{\pi}')\delta$ represents a strategy following the sequence of steps from π_1 to π_n but, as it is about to consume step π_n, gets "hijacked" by a transition in a $P_j \xrightarrow{\tilde{\mu}_j} P'_j$ sequence from Definition 6.4.4. The $\tilde{\pi}'$ part is a sequence of steps forced by that sequence and is such that its last step prevents π_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 δ 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 port (whose activeness is being proved) is respectively the input or the output at $\mathfrak{l}$'s n^{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 ρ to consume $\mathfrak{l}_0$ and thereby bring $\mathfrak{l}$ to top-level, then consume $\mathfrak{l}$, to obtain activeness on its parameter port p").

Example A.5.6 *Consider the following process:*

$$P = !\,t^{l_t} \mid !\,a(x)^{l_a}.\bar{t}^{l_{\bar{t}}}.\bar{x}^{l_{\bar{x}}} \mid \overline{a}\langle b\rangle^{l_{\bar{a}}}.b^{l_b}.c^{l_c}.\bar{s}^{l_{\bar{s}}}$$

which is an annotated form of $\mathrm{ran}(P) = !\,t \mid !\,a(x).\bar{t}.\bar{x} \mid \overline{a}\langle b\rangle.b.c.\bar{s}$.

The strategy for $\bar{s}_{\mathbf{A}} \triangleleft \bar{c}_{\mathbf{A}}$ *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* $\bar{s}_{\mathbf{A}} \triangleleft \bar{c}_{\mathbf{A}} : \rho$*). 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* τ_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* $!\,a(x).\bar{t}.\bar{x}$ *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* τ*-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* τ*-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 activeness proof.*

In this particular case, if a is input plain, the dependency statement becomes $\bar{s}_{\mathbf{A}} \triangleleft (\bar{c}_{\mathbf{A}} \wedge a_{\mathbf{R}})$*, which can be written* $(\bar{s}_{\mathbf{A}} \triangleleft \bar{c}_{\mathbf{A}}) \vee (\bar{s}_{\mathbf{A}} \triangleleft a_{\mathbf{R}})$*. The strategy for* $\bar{s}_{\mathbf{A}} \triangleleft a_{\mathbf{R}}$ *is*

$$(l_{\bar{a}}|l_a) \not\downarrow_t (l_{\bar{a}}|\bullet).\big((l_b|(\bullet|l_{\bar{a}})[\,\overline{1}\,]].(l_c|\bullet].\, l_{\bar{s}}\big) :$$

If a transition sequence $\tilde{\mu}_i$ *from (6.4.4) 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* $\not\downarrow_t$ *is followed, doing a* labelled *transition* $\xrightarrow{b}$ *instead of* $\xrightarrow{\tau_b}$*. The b-output, communication partner of* l_b*, is obtained with* $(\bullet|l_{\bar{a}})[\,\overline{1}\,]$*.*

Note that the strategy $(l_{\bar{a}}|\bullet).\big((l_b|(\bullet|l_{\bar{a}})[\,\overline{1}\,]].(l_c|\bullet].\, l_{\bar{s}}\big)$ *on its own corresponds to the statement* $\bar{s} \triangleleft a_{\mathbf{AR}}$ *— the strategy itself decided to do a labelled transition* $\xrightarrow{\overline{a}\langle b\rangle}$*, and therefore requires activeness on a from the environment.*

The following example shows more clearly how activeness 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* $\bar{s}_{\mathbf{A}} \triangleleft \bigvee_{i\in I} \bigwedge_{j\in I} (\bar{t}_{i\,\mathbf{A}} \wedge \bar{u}_{j\,\mathbf{A}})$ *or, equivalently,*

$$\bigvee_{j\in I} \bigwedge_{i\in I} \big(\bar{s}_{\mathbf{A}} \triangleleft (\bar{t}_{i\,\mathbf{A}} \wedge \bar{u}_{j\,\mathbf{A}})\big) \tag{105}$$

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 ρ_{ij} into the behavioural statement (105) gives the following annotated statement *for P:*

$$\bigvee_{j\in I}\bigwedge_{i\in I}\big(\bar{s}_{\mathbf{A}} \triangleleft \big(\bar{t}_{i\mathbf{A}} \wedge \bar{u}_{j\mathbf{A}}\big) : \big((l_{ti}|\bullet].\,(l_{ai}|l_{\bar{a}})\lightning\big((l_{tj}|\bullet].\,(l_{aj}|l_{\bar{a}})\big).\,(l_{uj}|\bullet].\,l_{xj}\big)\big)$$

We will often set conditions on strategies *and strategies they contain.* The following definition makes that concept precise.

Definition A.5.7 (Sub-strategies) *The* contains *relation is the least transitive relation on activeness 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 $1 \leq i < n$, ρ contains ρ_i and $\pi_1.\,\cdots.\,\mathfrak{l}_i$.*
- *Let $\rho = \pi_1.\,\cdots.\,(\mathfrak{l}_n|\rho_n)\lightning(\tilde{\pi}')\delta$. Then ρ contains $(\pi_1.\,\cdots.\,\mathfrak{l}_n)$, ρ_n and $\tilde{\pi}'\delta$.*
- *Let $\rho = (\bullet|\rho_0)\,[\,s\,]$. Then ρ contains ρ_0.*

If ρ contains ρ_0, the latter is called a sub-strategy *of the former.*

Just like activeness strategies prove correctness of an activeness 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 A.5.8 (Responsiveness Strategy) *A* responsiveness strategy *is an expression $\rho.\,\phi$ where ϕ is generated by the following grammar:*

$$\phi \;::=\; s_{\mathbf{A}} : \rho \;\;\big|\;\; p_{\mathbf{R}} : \rho.\,\phi \;\;\big|\;\; p_{\mathbf{R}} : \bullet \;\;\big|\;\; \phi \vee \phi \;\;\big|\;\; \phi \wedge \phi \;\;\big|\;\; \top \;\;\big|\;\; \bullet$$

where p ranges over numerical ports and s over sums of numerical ports.

Definition A.5.9 (Annotated Responsiveness Statement)
Removing Annotations *of a responsiveness strategy ϕ (in which $\bullet$ may only appear behind a "γ:" prefix), written* $\mathrm{ran}(\phi)$*, is the logical homomorphism yielding a dependency ε such that*

- $\mathrm{ran}(s_{\mathbf{A}} : \rho) \stackrel{\text{def}}{=} s_{\mathbf{A}}$
- $\mathrm{ran}(p_{\mathbf{R}} : \rho.\,\phi) \stackrel{\text{def}}{=} p_{\mathbf{R}}$

Let p be a port and ξ 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 $p_{\mathbf{R}} \lhd \varepsilon \colon \rho.\, \phi$, ρ tells precisely which p-prefix is being talked about, and ϕ 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.

Activeness and responsiveness strategies can be put together as follows:

Definition A.5.10 (Annotated Behavioural Statement)

Annotated behavioural statements *(ranged over by Φ) follow the grammar for Ξ given in Definition 6.2.1 on page 29, but where the $\gamma \lhd \varepsilon$ rule is replaced by annotated statements*

$$\cdots \quad \big| \quad s_{\mathbf{A}} \lhd \varepsilon \colon \rho \quad \big| \quad p_{\mathbf{R}} \lhd \varepsilon \colon \rho.\, \phi \quad \big| \quad \cdots$$

Definition A.5.11 (Annotated Process Type) *An annotated process type is a structure of the form $(\Sigma; \Phi_{\mathrm{L}} \blacktriangleleft \Xi_{\mathrm{E}})$ where Φ_{L} is an annotated behavioural statement, and Ξ_{E} a behavioural statement.*

Removing strategy annotations from an annotated process type is again done by the ran *operator, that recursively replaces $s_{\mathbf{A}} \lhd \varepsilon \colon \rho$ and $p_{\mathbf{R}} \lhd \varepsilon \colon \rho.\, \phi$ by $s_{\mathbf{A}} \lhd \varepsilon$ and $p_{\mathbf{R}} \lhd \varepsilon$, respectively.*

A.5.3 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 A.5.4 requires to have a strategy for every possible interference.

The sub operator gives the port brought to top-level by the given strategy, the 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 π.

For instance, having $P = a(x)^{l_a}.\overline{b}\langle x \rangle^{l_b} \,|\, \overline{a}\langle t \rangle^{l_{\bar{a}}}$:
$\mathsf{sub}_P((l_a|l_{\bar{a}})\,.\,l_{\bar{b}}) = \overline{b}$, $\mathsf{obj}_P((l_a|l_{\bar{a}})\,.\,(l_{\bar{b}}|\bullet)) = t$, 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 $P = \,!\, a^{l_a}.(\boldsymbol{\nu} n)\, Q \,|\, \overline{a}^{l_1}.Q_1 \,|\, \overline{a}^{l_2}.Q_2$, 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 $\overline{\mathfrak{p}}$ of such an extended port is obtained with $\overline{\pi.\,\mathfrak{p}} \stackrel{\text{def}}{=} \pi.\,\overline{\mathfrak{p}}$ and $\overline{\boldsymbol{\nu} p} \stackrel{\text{def}}{=} \boldsymbol{\nu}\bar{p}$.

Definition A.5.12 (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 ρ 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 $\overline{\mathsf{subst}_P(\tilde{\pi})a}$ if $\mathfrak{p} = \bar{a}$. Finally, it acts on each extended port individually when passed a tuple as in $\mathsf{subst}_P(\tilde{\pi})\tilde{\mathfrak{p}}$.*

These three functions are defined inductively on ρ, according to the following rules:

1. $\mathsf{sub}_P(\mathfrak{l}) \stackrel{\text{def}}{=} (\boldsymbol{\nu}\tilde{z})\,\mathsf{sub}(G)$ *(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. $\mathsf{sub}_P(\pi\,[\,p\,]) \stackrel{\text{def}}{=} \mathsf{obj}_P(\bar{\pi})\,[\,p\,]$
   (where $(\mathfrak{x}_1, \ldots, \mathfrak{x}_n)\,[\,i\,] = \mathfrak{x}_i$ and $(\mathfrak{x}_1, \ldots, \mathfrak{x}_n)[\,\bar{i}\,] = \overline{\mathfrak{x}_i}$)
4. $\mathsf{sub}_P(\tilde{\pi} \not\downarrow (\tilde{\pi}')\delta) \stackrel{\text{def}}{=} \mathsf{sub}_P(\tilde{\pi}'\delta)$
5. $\mathsf{obj}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho)) \stackrel{\text{def}}{=} \mathsf{subst}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho))\mathsf{obj}(G)$.
6. $\mathsf{subst}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho))a \stackrel{\text{def}}{=} a$ *if $a \notin (\mathrm{bn}(G) \cup \tilde{z})$*
7. $\mathsf{subst}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho))a \stackrel{\text{def}}{=} \tilde{\pi}.\,\boldsymbol{\nu} a$ *if $a \in \tilde{z}$*
8. $\mathsf{subst}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho))(\mathsf{obj}(G)\,[\,i\,]) \stackrel{\text{def}}{=} \mathsf{obj}_P((\rho|\tilde{\pi}.\,\mathfrak{l}))\,[\,i\,]$ *if G is an input and if $\rho \neq \bullet$.*
9. $\mathsf{subst}_P(\tilde{\pi}.\,\pi)a \stackrel{\text{def}}{=} \tilde{\pi}.\,\pi.\,\boldsymbol{\nu} a$ *if $a \in \mathrm{bn}(G)$ and either $\pi = (\mathfrak{l}|\rho]$ for some ρ, or $\pi = (\mathfrak{l}|\bullet)$, or G is an output and $\pi = (\mathfrak{l}|\rho)$ for some ρ.*
10. *All operators used in this definition commute with sums, so for example*

$$\mathsf{sub}_P(\sum_i \mathfrak{l}_i) \stackrel{\text{def}}{=} \sum_i \mathsf{sub}_P(\mathfrak{l}_i)$$

$$(\boldsymbol{\nu}\tilde{z}) \sum_i \mathfrak{p}_i \stackrel{\text{def}}{=} \sum_i (\boldsymbol{\nu}\tilde{z})\,\mathfrak{p}_i$$

$$\mathsf{sub}_P(\pi \left[\sum_i \mathfrak{l}_i \right]) \stackrel{\text{def}}{=} \sum_i \mathsf{sub}_P(\pi\,[\,\mathfrak{l}_i\,])$$

We omit the index P when there is no ambiguity.

As a first step to deciding correctness of a strategy, the following definition tells whether a strategy for an activeness resource $p_{\mathbf{A}}$ is actually able to bring p 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 $(l_1|\rho_1).(l_2|\rho_2).l_3$, ρ_1 could interfere with ρ_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 $\Sigma(a) = \langle\tilde{\sigma};\xi_{\mathrm{I}};\xi_{\mathrm{O}}\rangle$ (Σ being Γ's channel type mapping). Then $\Sigma_P(\rho)$ is ξ_{I} if $p = a$ and ξ_{O} if $p = \bar{a}$.

Definition A.5.13 (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 A.5.5.*
- *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 A.5.5.*
  - *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 $\tilde{\pi}.(\mathfrak{l}|\rho)\not\downarrow(\tilde{\pi}'.(\mathfrak{l}'|\rho'))\delta$, $\tilde{\pi}.(\mathfrak{l}|\rho)$ 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 sub. The substitution $\mathsf{subst}(\pi)$ is only applied to subsequent objects and subjects when π is doubly anchored (cf. Definition A.5.12). Therefore, in process (104) on page 91, 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 ρ with subject p is runnable then there is some ε such that $p_{\mathbf{A}} \triangleleft \varepsilon : \rho$ is correct in absence of interference. The following definition gives a lower bound (proven as a part of Lemma A.5.31) on ε:

Definition A.5.14 (Dependencies of a Strategy) *Let P be a process and ρ a runnable strategy. Then ρ's* dependencies *wrt. P (written $\mathsf{dep}_P(\rho)$) is the dependency ε defined as follows.*

- $\mathsf{dep}_P(\mathfrak{s}) \stackrel{\text{def}}{=} \top$
- $\mathsf{dep}_P((\bullet|\rho)\,[\,s\,]) \stackrel{\text{def}}{=} \overline{\mathsf{sub}_P(\rho)}_{\mathbf{AR}}$

- $\mathsf{dep}_P((\mathfrak{l}|\bullet)) \stackrel{\text{def}}{=} \mathsf{dep}_P((\mathfrak{l}|\bullet]) \stackrel{\text{def}}{=} \overline{\mathsf{sub}_P(\mathfrak{l})}_{\mathbf{A}}$
- $\mathsf{dep}_P((\mathfrak{l}|\rho)) \stackrel{\text{def}}{=} \mathsf{dep}_P((\mathfrak{l}|\rho]) \stackrel{\text{def}}{=} \mathsf{dep}_P(\rho)$
- $\mathsf{dep}_P(\pi.\,\rho) \stackrel{\text{def}}{=} \mathsf{dep}_P(\pi) \wedge \mathsf{dep}_P(\rho)$.
- $\mathsf{dep}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho)\lightning(\tilde{\pi}').\,\rho_2) \stackrel{\text{def}}{=} \mathsf{dep}_P(\tilde{\pi}.\,\mathfrak{l}) \wedge \mathsf{dep}_P(\rho_2)$
- $\mathsf{dep}_P(\tilde{\pi}.\,(\mathfrak{l}|\rho)\lightning(\bullet|\rho')\,[\tilde{p}]) \stackrel{\text{def}}{=} \mathsf{dep}_P(\tilde{\pi}.\,\mathfrak{l}) \wedge \overline{\mathsf{sub}_P(\rho')}_{\mathbf{R}}$

In the next to last point, $\tilde{\pi}'$ is irrelevant when computing a strategy's dependencies. The reason is that 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 only requires remote *responsiveness*, not activeness, 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 ϕ to closely follow the structure of the behavioural statement ξ.

Definition A.5.15 (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 ξ. We define $\tilde{\sigma}_i$ and ξ_q so that $\sigma_i = \langle \tilde{\sigma}_i; \xi_i; \xi_{\bar{\imath}} \rangle$.*

The dependencies of a responsiveness strategy ϕ for p, written $\mathsf{rdep}_P(\tilde{\sigma}, \xi, \phi)$, is inductively obtained as follows:

- $\mathsf{rdep}_P(\tilde{\sigma}, \top, \top) = \top$
- $\mathsf{rdep}_P(\tilde{\sigma}, \xi_1 \vee \xi_2, \phi_1 \vee \phi_2) = \mathsf{rdep}_P(\tilde{\sigma}, \xi_1, \phi_1) \vee \mathsf{rdep}_P(\tilde{\sigma}, \xi_2, \phi_2)$
- $\mathsf{rdep}_P(\tilde{\sigma}, \xi_1 \wedge \xi_2, \phi_1 \wedge \phi_2) = \mathsf{rdep}_P(\tilde{\sigma}, \xi_1, \phi_1) \wedge \mathsf{rdep}_P(\tilde{\sigma}, \xi_2, \phi_2)$
- $\mathsf{rdep}_P(\tilde{\sigma}, s_{\mathbf{A}} \triangleleft \varepsilon, s_{\mathbf{A}} : \rho) = \mathsf{dep}_P(\rho) \setminus \varepsilon$
- $\mathsf{rdep}_P(\tilde{\sigma}, q_{\mathbf{R}} \triangleleft \varepsilon, q_{\mathbf{R}} : \rho.\,\phi) = \mathsf{rdep}_P(\tilde{\sigma}_{\mathrm{n}(q)}, \xi_q, \phi) \setminus \varepsilon$
- $\mathsf{rdep}_P(\tilde{\sigma}, \gamma \triangleleft \varepsilon, \bullet) = \gamma$

Runnability is lifted to process types, by requiring each of its strategies to be runnable and respect the declared dependencies:

Definition A.5.16 (Consistent Typed Process) *An annotated typed process $(\Gamma; P)$ is said* consistent *if*

- *for every activeness statement $s_{\mathbf{A}} \triangleleft \varepsilon : \rho$ in Γ's local component, ρ is runnable for P, $\mathsf{dep}_P(\rho) \succeq \varepsilon$ and $\mathsf{sub}_P(\rho) = s$ (in case s is a sum, up to reordering of its terms).*
- *for every responsiveness statement $p_{\mathbf{R}} \triangleleft \varepsilon : \rho.\,\phi$ in Γ's local component, for every activeness strategy ρ' appearing in ϕ, $(\rho|\bullet).\,\rho'$ is runnable for P and $\mathsf{rdep}_P(\tilde{\sigma}_{\mathrm{n}(p)}, \xi_p, \phi) \succeq \varepsilon$, writing $\langle \tilde{\sigma}_a; \xi_a; \xi_{\bar{a}} \rangle$ for the type of a channel a in Γ.*

A.5.4 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 π-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

$$(\boldsymbol{\nu} a)\,(a(x)^{l_a}.\bar{x}.\bar{s} \mid \overline{a}\langle b\rangle^{l_b} \mid \overline{a}\langle c\rangle^{l_c} \mid P) \tag{106}$$

(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 activeness strategy $\tilde{\pi}.\,(\mathfrak{l}|\rho)\delta$ where $\mathfrak{l}$ could find partners ρ_i other than ρ. There should then be as many $\tilde{\pi}.\,(\mathfrak{l}|\rho)\not\downarrow(\mathfrak{l}|\rho_i)\delta_i$, 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 Γ_i corresponds to one term of the sum. We identify one particular choice with the corresponding event l_i:

Definition A.5.17 (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* ρ (called a *selection strategy*). For instance in

$$Q = \mathbf{!}\,a(\tilde{y})^{l_a}.P \,|\, \overline{a}\langle\tilde{x}\rangle^{l} \,|\, \overline{a}\langle\tilde{z}\rangle^{l'} \tag{107}$$

where P is as in Definition A.5.17, 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 (106) 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 (107) 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 A.5.18 (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* π_i *and* π'_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 ρ-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 A.5.19 (Contradicting Strategies) *Let* P *be a process.*

Two strategies ρ_1 *and* ρ_2 contradict *wrt.* 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*,* ρ_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* ρ_1 *doesn't match* ρ_2*, and, for both* i*,* ρ_i *contains* $\tilde{\pi}_i. (\mathfrak{l}|\rho'_i)$*.*

Remember (Definition A.5.7) that a strategy $\rho = (\mathfrak{l}|\rho_0) \not\downarrow (\mathfrak{l}|\rho_1). \rho'$ doesn't contain $(\mathfrak{l}|\rho_0)$ but does contain $(\mathfrak{l}|\rho_1). \rho'$. So (assuming ρ_0 and ρ_1 don't match) ρ contradicts $(\mathfrak{l}|\rho_0)$ but not $(\mathfrak{l}|\rho_1)$, as ρ_1 is $\mathfrak{l}$'s actual communication partner, although the strategy was "planning" to use ρ_0.

Definition A.5.20 (Choice Set) *Let* P *be an annotated process. A* choice set *for* P *is a finite set of runnable activeness 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 activeness 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 A.5.21 (Completeness) *An annotated behavioural statement* Φ *is* complete *with respect to* P *if, having* $\Phi \cong \bigvee_{i \in I} \Phi_i$*, 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* $\tilde{\rho}_C$*.*

A.5.5 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

89. Essentially, it is an activeness 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 A.5.22 (Strategy Marking Operator) *Let P be an annotated process and $\mathfrak{l}$ an extended event.* Marking P *with* $\mathfrak{l}$*, written* $\mathsf{mark}_{\mathfrak{l}}(P)$*, produces the annotated process inductively defined as follows:*

- $\mathsf{mark}_{\mathfrak{l}}(l) \stackrel{\text{def}}{=} \mathfrak{l}.\, l$
- $\mathsf{mark}_{\mathfrak{l}}(\mathfrak{l}_1.\, \mathfrak{l}_2) \stackrel{\text{def}}{=} \mathfrak{l}_1.\, \mathsf{mark}_{\mathfrak{l}}(\mathfrak{l}_2)$
- $\mathsf{mark}_{\mathfrak{l}}(G^{\mathfrak{l}'}.P) \stackrel{\text{def}}{=} G^{\mathsf{mark}_{\mathfrak{l}}(\mathfrak{l}')}.\mathsf{mark}_{\mathfrak{l}}(P)$
- $\mathsf{mark}_{\mathfrak{l}}(P_1|P_2) \stackrel{\text{def}}{=} \mathsf{mark}_{\mathfrak{l}}(P_1) \,|\, \mathsf{mark}_{\mathfrak{l}}(P_2)$
- $\mathsf{mark}_{\mathfrak{l}}(P_1{+}P_2) \stackrel{\text{def}}{=} \mathsf{mark}_{\mathfrak{l}}(P_1) + \mathsf{mark}_{\mathfrak{l}}(P_2)$
- $\mathsf{mark}_{\mathfrak{l}}((\boldsymbol{\nu}\mathfrak{a})\, P) \stackrel{\text{def}}{=} (\boldsymbol{\nu}\mathsf{mark}_{\mathfrak{l}}(\mathfrak{a}))\, (\mathsf{mark}_{\mathfrak{l}}(P)\{^{\mathsf{mark}_{\mathfrak{l}}(\mathfrak{a})}/_{\mathfrak{a}}\})$
- $\mathsf{mark}_{\mathfrak{l}}(\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0}$

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 τ-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.

$$\frac{-}{\overline{a}\langle\tilde{x}\rangle^{\mathfrak{l}}.P \xrightarrow{\overline{a}\langle\tilde{x}\rangle,(\mathfrak{l}|\mathfrak{l}')} \mathsf{mark}_{\mathfrak{l}'}(P)} \text{ (A-Out)}$$

$$\frac{-}{a(\tilde{y})^{\mathfrak{l}}.P \xrightarrow{a(\tilde{x}),(\mathfrak{l}|\mathfrak{l}')} \mathsf{mark}_{\mathfrak{l}'}(P)\{^{\tilde{x}}/_{\tilde{y}}\}} \text{ (A-Inp)}$$

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\tilde{z}:\tilde{\sigma})\,\overline{a}\langle\tilde{x}\rangle,(\mathfrak{l}_o|\mathfrak{l}_i)} P' \quad Q \xrightarrow{a(\tilde{x}),(\mathfrak{l}_i|\mathfrak{l}_o)} Q'}{\begin{array}{l} P \,|\, Q \xrightarrow{\tau,(\mathfrak{l}_i|\mathfrak{l}_o)} (\boldsymbol{\nu}\tilde{z}:\tilde{\sigma})\,(P' \,|\, Q') \\ Q \,|\, P \xrightarrow{\tau,(\mathfrak{l}_i|\mathfrak{l}_o)} (\boldsymbol{\nu}\tilde{z}:\tilde{\sigma})\,(Q' \,|\, P') \end{array}} \text{ (A-Com)}$$

Rules (A-Open), (A-Rep), (A-New), (A-Par), (A-Sum) and (A-Cong) are identical to the corresponding ones in Table 2 on page 5 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, activeness strategies need to be updated accordingly:

Definition A.5.23 (Strategy Transition Operator) *Let ρ be an activeness strategy and π an event pair $(\mathfrak{l}_1|\mathfrak{l}_2)$ where $\mathfrak{l}_2$ may be $\bullet$.*

Then $\rho \wr \pi$ is the activeness strategy obtained as follows (the word "otherwise" is used in the sense "if none of the previous rules apply").

- *If ρ and π contradict then $\rho \wr \pi = \bot$.*
- $\mathfrak{l} \wr \pi \stackrel{\text{def}}{=} \mathfrak{l}$ *otherwise.*
- *If one of $\pi.\,\rho_0$ and $\bar{\pi}.\,\rho_0$ matches $\pi_0.\,\rho_0$ then $(\pi_0.\,\rho_0) \wr \pi \stackrel{\text{def}}{=} \mathsf{mark}_{\mathfrak{l}_2}(\rho_0) \wr \pi$.*
- $((\mathfrak{l}_0|\rho_0).\,\rho_1) \wr \pi \stackrel{\text{def}}{=} (\mathfrak{l}_0|\rho_0 \wr \pi).\,(\rho_2 \wr \pi)$ *otherwise.*
- $\pi_0\,[\,\tilde{q}\,] \wr \pi = \bot$ *if π or $\bar{\pi}$ matches π_0.*
- $(\bullet|\rho_0)\,[\,\tilde{q}\,] \wr \pi = (\bullet|\rho_0 \wr \pi)\,[\,\tilde{q}\,]$ *otherwise.*
- $(\tilde{\pi} \lightning (\pi)\delta) \wr \pi \stackrel{\text{def}}{=} (\tilde{\pi} \lightning (\pi)\delta) \wr \bar{\pi} \stackrel{\text{def}}{=} (\pi\delta) \wr \pi$.
- $\big(\tilde{\pi} \lightning (\tilde{\pi}')\delta\big) \wr \pi \stackrel{\text{def}}{=} (\tilde{\pi} \wr \pi) \lightning \big((\tilde{\pi}')\delta \wr \pi\big)$ *otherwise.*

with the following extension of the mark *operator from Definition A.5.22:*

- $\mathsf{mark}_{\mathfrak{l}}((\mathfrak{l}_0|\rho_0)) \stackrel{\text{def}}{=} (\mathsf{mark}_{\mathfrak{l}}(\mathfrak{l}_0)|\rho_0)$.
- $\mathsf{mark}_{\mathfrak{l}}(\tilde{\pi} \lightning (\rho_0)\delta) \stackrel{\text{def}}{=} \mathsf{mark}_{\mathfrak{l}}(\tilde{\pi}) \lightning (\rho_0)\delta$

That operator is lifted to behavioural statements: $\Phi \mapsto \Phi \wr \pi$ is a logical homomorphism such that

- *If $\Phi \cong \top$ then $\Phi \wr \pi \stackrel{\text{def}}{=} \top$.*
- *If $\rho \wr \pi = \bot$ then*
  - $(s_{\mathbf{A}} \lhd \varepsilon\colon \rho) \wr \pi \stackrel{\text{def}}{=} \bot$
  - $(p_{\mathbf{R}} \lhd \varepsilon\colon \rho.\,\phi) \wr \pi \stackrel{\text{def}}{=} \top$
- *otherwise,*
  - $(s_{\mathbf{A}} \lhd \varepsilon\colon \rho) \wr \pi \stackrel{\text{def}}{=} s_{\mathbf{A}} \lhd \varepsilon\colon (\rho \wr \pi)$
  - $(p_{\mathbf{R}} \lhd \varepsilon\colon \rho.\,\phi) \wr \pi \stackrel{\text{def}}{=} p_{\mathbf{R}} \lhd \varepsilon\colon (\rho \wr \pi).\,(\phi \wr \pi)$ *(where $\phi \wr \pi$ follows the same rules as $\Phi \wr \pi$, without the ε)*
- $\Phi \wr \pi \stackrel{\text{def}}{=} \Phi$ *when no other rules apply.*

On process types, $(\Sigma; \Phi_{\mathrm{L}} \blacktriangleleft \Phi_{\mathrm{E}}) \wr \pi \stackrel{\text{def}}{=} (\sigma; \Phi_{\mathrm{L}} \wr \pi \blacktriangleleft \Phi_{\mathrm{E}})$.

Transition on annotated typed processes are defined similarly to those on typed processes in Definition 3.2.4:

Definition A.5.24 (Typed Labelled Transition System) *The* Transition Operator $\Gamma \wr \mu$ *on annotated process types modifies the type precisely as in Definition 6.3.12 on page 37.*

The labelled transition relation on annotated typed processes:

$$(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$$

if there is π *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 6.3.12 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 6.4.4 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 1. 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 A.5.25 (LTS Equivalence) *Let* $(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'; P')$ *be a transition sequence on annotated typed processes. Then* $\mathrm{ran}(\Gamma; P) \xrightarrow{\tilde{\mu}} \mathrm{ran}(\Gamma'; P')$.

Reciprocally, let $\mathrm{ran}(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'; P')$ *be a transition sequence on* non-*annotated typed processes. Then there is exactly one* $(\Gamma'_0; P'_0)$ *with* $(\Gamma; P) \xrightarrow{\tilde{\mu}} (\Gamma'_0; P'_0)$ *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 A.5.26 (Subject Transitions) *Let* $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ *be a transition on annotated typed process and let* π *be the strategy step used to prove it using Definition A.5.24, and let* $\mathfrak{p}$ *an extended port. Then there is an extended port* $\mathfrak{p}'$ *such that, for any runnable strategy* ρ *such that* $\rho \wr \pi \neq \bot$, $\mathsf{sub}_P(\rho) = \mathfrak{p}$ *implies* $\mathsf{sub}_{P'}(\rho \wr \pi) = \mathfrak{p}'$, *and* $\mathsf{sub}_P(\rho) = \bar{\mathfrak{p}}$ *implies* $\mathsf{sub}_{P'}(\rho \wr \pi) = \overline{\mathfrak{p}'}$.

Proof 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 ρ 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 $\rho' \neq \bot$, ρ and π 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 π_0 matches π.

3. $\rho = \tilde{\pi}.\mathfrak{l}$ and π_0 does not match π.

   As π and π_0 do not match, by non-contradiction, the $\mathfrak{l}_0$-guard must still be available unchanged in P' (up to α-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 π_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 α-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 α-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 π_0 matches π.

   Let $\bar{\mathfrak{l}}_0$ be such that $\{\mathfrak{l}_0, \bar{\mathfrak{l}}_0\} = \{\mathfrak{l}_I, \mathfrak{l}_O\}$. Then ρ is transformed into ρ' as follows: The π_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 ρ_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 ρ 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. $\rho = \tilde{\pi}.\mathfrak{l}$, π_0 matches π, 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. $\rho = \tilde{\pi}.\,\mathfrak{l}$, π_0 matches π, 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 α-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 substitution works as before and we get $\mathsf{sub}_{P'}(\pi'_1.\,\cdots.\,\hat{\mathfrak{l}}) = p$, as required.

7. $\rho = \tilde{\pi}.\,\mathfrak{l}$, π_0 matches π, and $\mathsf{sub}_P(\pi_1.\,\cdots.\,\hat{\mathfrak{l}}) = q \in \mathrm{bn}(G)$.

   In this case q got substituted to p by π_0. This requires (Definition A.5.12) π_0 to be doubly anchored, which in turn requires (for π_0 to match π) $\pi_0 = (\mathfrak{l}_0|\bar{\mathfrak{l}}_0)$. $\mathsf{subst}_P(\pi_0)$ 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 μ is a τ or an input then $\mathfrak{p}' = \mathfrak{p}$. If μ 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 A.5.12, $\mathsf{sub}_{(\boldsymbol{\nu}\tilde{z})\,P_0}(\rho) = (\boldsymbol{\nu} p)$ 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 μ did some α-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 μ's objects given above holds.

Now let $\mathfrak{p} = \hat{\pi}.\,\boldsymbol{\nu} p$, where $\hat{\pi} = (\hat{\mathfrak{l}}|\hat{\rho})$ matches π. 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 α-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 A.5.12, 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 A.5.12, $\mathsf{sub}_P(\rho_j.\cdots.\mathfrak{l}) = \pi_j.\boldsymbol{\nu} q$, and then all subsequent π_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 π 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 π.

   The process P, as ρ 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 (π 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 104 (for $\mathfrak{p}$ free) applies here as well: $\mathsf{sub}_P(\mathfrak{l})$ gets replaced by $\hat{\pi}.\boldsymbol{\nu} p$ which becomes $q\{{}^{\mathsf{obj}(\mu)}/_{\mathsf{obj}(G)}\}$ after the transition, by induction hypothesis.

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 π_0^* that either does not match π, 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 A.5.23 and mark leaves bound names $\boldsymbol{\nu} p$ unchanged (up to α-renaming — the last π_j^* uniquely identifies in the process a binder of $\mathrm{n}(p)$, and if the transition α-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 π_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* π_j is not a doubly-anchored input. π_0 doesn't match π.

   As $\mathsf{sub}(\rho)$ binds p rather than substituting it, $\mathsf{sub}_P(\mathfrak{l}) = p$. As π doesn't match π_0 but doesn't contradict ρ, the $\mathfrak{l}_0$-guard G and its continuation Q are left unchanged by the transition, up to α-renaming, in particular the events $\mathfrak{l}_i$ are left as is. Let $\mathsf{sub}_{P'}(\mathfrak{l}) = p'$.

   As π_0 and π 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 π'_i with $i > j$. It is bound (not substituted) by π'_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* π_j is not a doubly-anchored input. π_0 matches π.

   As in point 3 of case $\mathfrak{p} = \boldsymbol{\nu} p$ on page 105, by ρ-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 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.
   - π_0^* doesn't match π. Directly contradicts "π_0 matches π".
   - $\mathrm{n}(p) \notin \mathrm{bn}(\mathfrak{l}_0^*)$. 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* π_j is not a doubly-anchored input. π_0 matches π. $j > 0$.

   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* π_j is not a doubly-anchored input. π_0 matches π. $\mathrm{n}(p) \notin \mathrm{bn}(\mathfrak{l}_0^*)$.

   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 α-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 π_0 matches π, $\pi_0.\,\boldsymbol{\nu} p' \wr \pi = \boldsymbol{\nu} p'$ so we are done.

□

The previous lemma implies (proved as part of Lemma A.5.29) 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 ρ' there is a corresponding P-runnable strategy ρ 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 A.5.27 (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 ρ' *such that* $\mathsf{sub}_{P'}(\rho') = \mathfrak{p}'$ *there is a strategy* ρ *that satisfies the guarding and top-levelness constraints of Definition A.5.13, 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).*

Proof The construction of ρ from ρ' 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 α-renaming). The "and/or" is resolved by checking if μ is an input (only produce Q'_I), an output (only Q'_O) or a τ (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 π_0 is "neutral" in the sense that $\pi_0.\,\rho \stackrel{\text{def}}{=} \rho$.

Apply this $\rho' \mapsto \rho$ transformation inductively (for the same transition μ and step π) to obtain ρ_i, for all i s.t. $\rho'_i \neq \bullet$. The remaining ρ_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 π_1 to π_{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 ρ_i is of the form $(\bullet|\mathfrak{l})\,[\,q\,]$ with $\mathfrak{l} \in \{\mathfrak{l}_I, \mathfrak{l}_O\}$.

The strategy ρ 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 ρ' was of the form $(\bullet|\rho'_0)\,[\,p\,]$, transform ρ'_0 into ρ_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 ρ 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 ρ 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 μ is an input its objects must be fresh. If μ 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 μ is an output and $\pi = (\mathfrak{l}_I|\bullet)$ in case μ 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 μ, i.e. G_I if μ 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 ρ 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 ρ_j satisfying the lemma conditions (where ρ' and ρ in the statement stand for ρ'_j and ρ_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 α-renaming occurred). Then $\mathrm{n}(p_0)$ is not bound by any of the prefixes corresponding to π_i with $j < i < n$ (as that property is preserved by α-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 μ'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 A.5.26, 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 α-renaming occurring in the $P \xrightarrow{\mu;\pi} P'$ transition and π_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 ρ' be a strategy with subject $\mathfrak{p}'$ and define π'_i, $\mathfrak{l}'_i$, ρ'_i and their counterparts without a tick $'$ as in all previous cases. Note that the π'_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 π'_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 π_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.

□

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 A.5.28 (Weight of a Strategy) *The* weight $\mathsf{wt}(\rho)$ *of a strategy* ρ *is defined inductively:*

- $\mathsf{wt}(\mathfrak{l}) \stackrel{\text{def}}{=} \mathsf{wt}(\bullet) \stackrel{\text{def}}{=} 0$
- $\mathsf{wt}((\bullet|\rho)\,[\,p\,]) \stackrel{\text{def}}{=} \mathsf{wt}((\mathfrak{l}|\rho)) \stackrel{\text{def}}{=} \mathsf{wt}((\mathfrak{l}|\rho]) \stackrel{\text{def}}{=} \mathsf{wt}(\rho) + 1$
- $\mathsf{wt}(\pi.\,\rho) \stackrel{\text{def}}{=} \mathsf{wt}(\pi) + \mathsf{wt}(\rho)$
- $\mathsf{wt}(\tilde{\pi}_1 \lightning (\tilde{\pi}_2)\delta) \stackrel{\text{def}}{=} \mathsf{wt}(\tilde{\pi}_1) + \mathsf{wt}(\tilde{\pi}_2\delta) - \mathsf{wt}(\tilde{\pi}_2)$

"Elementary" in the following definition refers to the image of relation $\searrow$. See Definition 6.4.2 on page 40.

Lemma A.5.29 (Runnability Safety) *Let* $(\Gamma; P)$ *be a consistent, complete and elementary annotated typed process. 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* $\mathsf{wt}(\Gamma) \leq \mathsf{wt}(\Gamma')$.

Proof First of all, Γ' is elementary by definition of the $\searrow$ relation.

We first prove Γ' is consistent before proceeding to completeness.

Let Γ's local dependency network be $s_{\mathbf{A}} \triangleleft \varepsilon : \rho$. Let $\pi = (\mathfrak{l}_I|\mathfrak{l}_O)$ be the step used to prove $(\Gamma; P) \xrightarrow{\mu}\!\searrow (\Gamma'; P')$ following Definition A.5.24. 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 p in Γ' is $\rho' = \rho\wr\pi$. We assume $\rho' \neq \bot$ otherwise $\Gamma' = \top$ which is vacuously consistent. This implies that π doesn't contradict ρ. By hypothesis, ρ is runnable. We show by induction on ρ's structure that all conditions in Definition A.5.13 are preserved in ρ'. By induction hypothesis, the strict substrategies of ρ' (i.e. those distinct from ρ' itself) 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 μ, 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 μ 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 π_0 matches π (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 A.5.26, $\mathfrak{p}' = p$ 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 A.5.26, $\mathsf{sub}_P(\hat{\rho}) = \bar{\mathfrak{p}}$ 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 $\rho = (\hat{\mathfrak{l}}|\hat{\rho}).\mathfrak{s}$. π_0 does not match π.

   ρ' is equal to ρ in the $\mathfrak{l}_i$, and the ρ_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 μ. In particular, the $\hat{\mathfrak{l}}$ guards $\mathfrak{s}$ condition is preserved.

6. $\rho = \tilde{\pi}.(\hat{\mathfrak{l}}|\hat{\rho}).\mathfrak{s}$ or $\rho = (\hat{\mathfrak{l}}|\hat{\rho}).\mathfrak{s}$. π_0 matches π.

   Let $\bar{\mathfrak{l}}_0$ be such that $\{\mathfrak{l}_0, \bar{\mathfrak{l}}_0\} = \{\mathfrak{l}_I, \mathfrak{l}_O\}$. Then ρ is transformed into ρ' as follows: The π_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 ρ_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 ρ by $\mathsf{mark}_{\bar{\mathfrak{l}}_0}(\mathfrak{l}_i)$. The two following cases cover the two possible forms of ρ.

7. $\rho = \tilde{\pi}.(\hat{\mathfrak{l}}|\hat{\rho}).\mathfrak{s}$. π_0 matches π.

   $\hat{\mathfrak{l}}$ 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 π.

   $\rho' = \mathfrak{l}'$. 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 A.5.27: Let ρ' be a selection strategy for P'. Then there is a selection strategy ρ for P such that $\rho \wr \pi = \rho'$.

The construction of ρ from ρ', μ and π is given in the proof of Lemma A.5.27. We show that ρ is runnable if ρ' 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 ρ'.

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 ρ' as given in the proof of Lemma A.5.27.

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 π_0 is "neutral", $\rho = \mathfrak{l}_1$ and there's nothing to show (we already showed as part of Lemma A.5.27 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 Γ' 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 ρ' is runnable, $\mathsf{sub}_{P'}(\pi'_1. \ldots \mathfrak{l}'_{n-1})$ is free in P' (let's call it p') and Γ'-observable.

   Applying Lemma A.5.27, $\mathfrak{p} = \mathsf{sub}_P(\pi_0. \pi_1. \ldots \mathfrak{l}_{n-1})$ 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 π, 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 A.5.27 we have $p = p'$ and p is necessarily observable in Γ as μ is either τ (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 A.5.27, $\mathfrak{p}$ 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 ρ_n and $\pi_0. \cdots . \mathfrak{l}_{n-1}$ have been obtained from ρ'_{n-1} and $\pi_1.{}' \cdots . \mathfrak{l}'_{n-1}$ following the same $\rho' \mapsto \rho$-construction as in Lemma A.5.27 we have $\mathsf{sub}_P(\rho_{n-1}) = \bar{\mathfrak{p}}$, as required.

□

A key component of proving correctness of an annotated process type is the following lemma, that effectively connects process structure (activeness strategies) and process behaviour (transition sequences).

Lemma A.5.30 (Strategy Application) *If* $(\Gamma; P)$ *is a consistent, complete and elementary annotated typed process, and* $\tilde{\rho}$ *be a choice set, then either* $(\Gamma; P)$ *is immediately correct or there is a transition* $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ *such that* $\mathsf{wt}(\Gamma') < \mathsf{wt}(\Gamma)$.

Proof 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 Γ's local dependency network be $s_{\mathbf{A}} \triangleleft \varepsilon : \rho$. 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 and $P \equiv (\boldsymbol{\nu}\tilde{a})\,(\sum_i G_i{}^{\mathfrak{l}_i}.Q_i \mid R)$, where $\sum_i \mathsf{sub}(G_i) = s$, so Γ is immediately correct.

If $\rho = (\rho_0 | \bullet)\,[\,s'\,]$, let $p_0 = \mathsf{sub}(\rho_0)$. Set Γ_0 to Γ but with local component $p_{0\mathbf{A}} \triangleleft \varepsilon : \rho_0$. As Γ is consistent and complete, so is Γ_0, and the induction hypothesis applies. Case 1: Γ_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 A.5.12 and consistency of Γ, $\mathsf{obj}(\mu)\,[\,s'\,] = s$, so $\Gamma \wr \mu$ drops activeness on s, 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 Γ_0' be $p_{0\mathbf{A}} \triangleleft \varepsilon' : \rho_0'$. Then the local component of Γ' is $s_{\mathbf{A}} \triangleleft \varepsilon' : (\rho_0' | \bullet)\,[\,s'\,]$. 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 ρ_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 μ' has $\bar{p}_0$ in subject position. Applying the (Com) rule of the LTS the μ' transition can be replaced by a τ-transition additionally consuming $\mathfrak{l}_0$, let that transition be $(\Gamma; P) \xrightarrow{\tau} (\Gamma'; P')$. Then the local component of Γ' is $s_{\mathbf{A}} \triangleleft \varepsilon : \rho_1$. If $\bar{p}_0$ is an object of the μ' transition, $\bar{p}_{0\mathbf{A}}$ will be provided by the environment and one can do (after μ') 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 Γ_0 be Γ but with local component $\bar{p}_{0\mathbf{A}} \triangleleft \varepsilon : \rho_0$. 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 Γ_0' being $p_{0\mathbf{A}} \triangleleft \varepsilon : \rho_0'$, we have $s_{\mathbf{A}} \triangleleft \varepsilon : (\mathfrak{l}_0 | \rho_0').\,\rho_1$ as local component of Γ'.

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 Γ' is ρ_1, which has a weight lower than ρ. By runnability of ρ and the definition of the dep_P operator, $\bar{p}_{0\mathbf{A}}$ must provided by the environment.

The $\rho = \tilde{\pi}_1 \lcurlyvee (\tilde{\pi}_2)\delta$ case is essentially identical to the above ones, by focusing on the $\tilde{\pi}_1$ part and leaving the rest unchanged.

□

Soundness of consistency and completeness follows as a simple corollary.

Corollary A.5.31 (Completeness and Correctness) *If* $(\Gamma; P)$ *is a consistent and complete annotated typed process, then* $\mathrm{ran}(\Gamma; P)$ *satisfies definition 6.4.4 for the special case where* $\tilde{\mu}_0$ *is empty.*

Proof 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 Γ'_0 has a weight lower or equal to that of Γ_0.

By repeated application of Lemma A.5.29 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 Γ'_i has a weight smaller than or equal to $\mathsf{wt}(\Gamma_i)$.

The strategy f is defined following Lemma A.5.30, 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 6.4.4 on page 41 of at most $\mathsf{wt}(\Gamma'_0)P'_0$ such that $i > n$ implies $(\Gamma_i; P_i)$ is immediately correct. □

A.5.6 Annotated Type System

Most of the soundness proof now amounts to lifting the type algebra and type system to work on annotated process types.

So in this section we 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 (R-Pre) rule constructs base strategies for the subject activeness and responsiveness, remote behaviour, and adds a transition at the beginning of all activeness strategies from the continuation.

The following definition tells how to reduce a $s_{\mathbf{A}} \triangleleft p_{\mathbf{A}} \triangleleft \varepsilon_p$ dependency chain:

Definition A.5.32 (Activeness-Activeness Reduction) *The reduction relation* $\hookrightarrow$ *is modifed as follows for annotated process types, in the context of a process* P*:*

Let $\Xi = (p_{\mathbf{A}} \triangleleft \varepsilon_p \colon \rho_p) \wedge (s_{\mathbf{A}} \triangleleft (p_{\mathbf{A}} \wedge \varepsilon_s) \colon \rho_s)$*. Then*

$$\Xi \hookrightarrow \Xi \ \wedge \ s_{\mathbf{A}} \triangleleft (\varepsilon_p \wedge \varepsilon_s) \colon \rho'_s$$

ρ'_s *is obtained from* ρ_s *by replacing as many sub-strategies as possible using the following rules, that each assume* $\mathsf{sub}_P(\tilde{\pi}.\, l) = \bar{p}$*.*

$$\tilde{\pi}.\, (l|\bullet).\, \rho \mapsto \tilde{\pi}.\, (l|\rho_p).\, \rho \tag{108}$$

$$\tilde{\pi}.\, (l|\bullet).\, \rho \mapsto \tilde{\pi}.\, (l|\rho_p).\, \rho \lcurlyvee \big(\tilde{\pi}.\, (l|\bullet)\big).\, \rho \tag{109}$$

$$(\bullet|\tilde{\pi}.\, \mathfrak{l})\,[\,r\,] \mapsto (\rho_r|\tilde{\pi}.\, \mathfrak{l}) \lcurlyvee (\bullet|\tilde{\pi}.\, \mathfrak{l})\,[\,r\,] \tag{110}$$

To reduce chains such as $s_{\mathbf{A}} \triangleleft p_{\mathbf{R}} \triangleleft \varepsilon_p$, one needs to convert a responsiveness strategy ϕ on parameter names into an activeness strategy, applying parameter instantiation to strategies:

Definition A.5.33 (Strategy Instantiation) *Let ϕ be an annotated responsiveness strategy and s a sum of parameter ports (n or $\bar{n}$). Then* instantiating *ϕ's port(s) s, written $\phi\,[\,s\,]$ is the logical homomorphism returning behavioural statements whose atoms are activeness strategies:*

- $(s'_{\mathbf{A}} : \rho)\,[\,s\,] = \rho$ *when* $s = s'$
- $\phi\,[\,s\,] = \top$ *when no other rules apply.*

Extracting an activeness strategy from a responsiveness strategy replaces the "unspecified" communication partner "$\bullet$" and parameter number "$[\,s\,]$" by an actual communication partner ρ_p and an instantiation of its responsiveness strategy $\phi\,[\,s\,]$.

Definition A.5.34 (Responsiveness-Activeness Reduction)
Let $\Xi = p_{\mathbf{R}} \triangleleft \varepsilon_p : \rho_p . \phi$ be an annotated behavioural statement for a process P. Then

$$\Xi \wedge (s_{\mathbf{A}} \triangleleft (p_{\mathbf{R}} \wedge \varepsilon_s) : \rho_s) \hookrightarrow \Xi \wedge \big((s_{\mathbf{A}} \triangleleft (p_{\mathbf{R}} \wedge \varepsilon_s) : \rho_s) \vee (s_{\mathbf{A}} \triangleleft (\varepsilon_p \wedge \varepsilon_s) : \rho'_s)\big) \quad (111)$$

where ρ'_s is obtained from ρ_s by repeatedly applying the following transformation on sub-strategies:

$$(\bullet|\rho_1)\,[\,s'\,] \mapsto (\bullet|\rho_1) \not\downarrow (\rho_p|\rho_1) . \phi\,[\,s'\,]$$

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 $p_{\mathbf{A}} \triangleleft \varepsilon : \rho'$ to $p_{\mathbf{A}} \triangleleft \varepsilon : (\rho|\bullet) . \rho'$ and $p_{\mathbf{R}} \triangleleft \varepsilon : \rho' . \phi$ to $p_{\mathbf{R}} \triangleleft \varepsilon : ((\rho|\bullet) . \rho') . \phi$.

Definition A.5.35 (Responsiveness Reduction) *Let Ξ be an annotated dependency statement and ϕ a responsiveness strategy such that $\Xi \wedge (\rho|\bullet) . \phi \hookrightarrow \Xi \wedge (\rho'|\bullet) . \phi'$ for some ϕ'. Then*

$$\Xi \wedge (p_{\mathbf{R}} \triangleleft \varepsilon : \rho . \phi) \hookrightarrow \Xi \wedge (p_{\mathbf{R}} \triangleleft \varepsilon' : \rho' . \phi')$$

where ε' is obtained as in Definition 6.3.4.

Gathering the above definitions together we obtain the annotated counterpart to Definition 6.3.4 on page 35. There are fewer cases because annotated process types only contain dependency statements on the local side.

Definition A.5.36 (Annotated Dependency Reduction)
The reduction relation $\hookrightarrow$ on annotated behavioural statements is a partial order relation satisfying

- *The reductions as given in Definitions A.5.32, A.5.34 and A.5.35.*
- $\Phi \hookrightarrow \Phi'$ *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 A.5.37 (Reduction Preserves Consistency) *If Φ is a consistent annotated behavioural statement for a process P and $\Phi \hookrightarrow \Phi'$ then Φ' is consistent as well.*

Proof We show that all four transformations (108), (109), (110) and (111) given in Definitions A.5.32 and A.5.34 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 $(s_{\mathbf{A}} \cdots : \rho) \mapsto (s_{\mathbf{A}} \cdots : \rho')$ implies $\mathsf{sub}(\rho) = \mathsf{sub}(\rho')$, i.e. $\mathsf{sub}(\rho') = s$ as is required for consistency.

(108) 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 (108). 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 ρ-component of a singly-anchored step and sub doesn't depend on such components, the subject of the resulting strategy is unchanged.

(109) As in the previous case the main event sequence is preserved by the transformation, and the complementarity of $\mathfrak{l}$ and ρ_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 ρ is the exact strategy prior to the transformation so we are done.

(110) The left hand side of the $\lightning$ symbol is runnable because both ρ_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.

(111) 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. □

Lemma A.5.38 (Composition Preserves Consistency) *Let Γ_1 and Γ_2 be annotated process types consistent for a process P. Then their composition $\Gamma_1 \odot \Gamma_2$ is consistent for P as well.*

Proof Following Definition 6.3.11 on page 37:

The first step simply combines into a single behavioural statement strategies from Γ_1 and Γ_2. As consistency of a statement is equivalent to runnability of all activeness 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 Γ_i doesn't imply it being observable in $\Gamma_1 \odot \Gamma_2$. Note however that those strategies violating the observability requirement all declare $\overline{\mathsf{sub}(\rho)}_{\mathbf{A}}$ in their dependencies, by consistency of behavioural statements.

The second step performs a number of dependency reductions which, by Lemma A.5.37, preserve consistency of the strategies, still disregarding the observability requirement.

Finally, the third step of Definition 6.3.11 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. □

When the $\hookrightarrow$ relation replaces some strategy ρ_0 by ρ, ρ_0 is a *precursor* of ρ. The two points in the list below respectively model transformations done by Definitions A.5.32 and A.5.34 on page 113.

Definition A.5.39 (Strategy Precursor) *An activeness strategy ρ_0 is said a* precursor *of a strategy ρ for some process P if ρ can be obtained from ρ_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 activeness strategies,*
- *replacing a sub-strategy $(\bullet|\rho_0)\,[\,q\,]$ by $\tilde{\pi}.\,(\mathfrak{l}|\rho_0).\,\rho'$*

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 A.5.40 (Pre-Completeness) *Let Φ be an annotated behavioural statement, P a process and $\tilde{\rho}$ be a choice set. Φ is said to be* pre-complete *for P with respect to $\tilde{\rho}$ if:*

- *No activeness strategy in Φ is self-contradicting.*
- *for any runnable activeness strategy ρ not contradicting $\tilde{\rho}$ and such that $\mathsf{sub}_P(\rho)$ is a free port p, Φ contains a statement $p_{\mathbf{R}} \triangleleft \varepsilon \colon \rho_0.\,\phi$ with ρ_0 being a precursor of ρ.*
- *for every annotated activeness statement $p_{\mathbf{A}} \triangleleft \varepsilon : \rho_2$ contained in Φ, for every runnable precursor ρ_1 of ρ_2, there is a precursor ρ_0 of ρ_1 such that Φ contains a statement $p_{\mathbf{A}} \triangleleft \varepsilon' \colon \rho_2$ for some ε'.*

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 Φ_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 Φ_1 and Φ_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 6.3.8 on page 37, 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 A.5.41 (Closure Completes) *If Γ is a consistent and pre-complete type for a process P, then Γ's closure is complete for P.*

Proof Let P and Γ be as in the statement, let $\bigvee_{i\in I}\Phi_i$ be the local component of close (Γ).

We show by induction on the size of a choice set $\tilde{\rho}$ that $\exists i \in I$ s.t. Φ_i doesn't contradict $\tilde{\rho}$, i.e. Φ is complete.

The base case ($\tilde{\rho} = \varnothing$) is immediate — if the choice set is empty it can't contradict any Φ_i.

Fix a choice set $\tilde{\rho}$. Let $I_0 \subseteq I$ be the set of i such that Φ_i doesn't contradict $\tilde{\rho}$. By induction hypothesis $I_0 \neq \varnothing$. Let ρ_c be a selection strategy such that $\tilde{\rho}\cup\{\rho_c\}$ is a choice set according to Definition A.5.20 (i.e. ρ_c is runnable, doesn't contradict any $\rho \in \tilde{\rho}$ and all proper sub-strategies of ρ_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 Φ_j doesn't contradict ρ_c.

Let $\hat{\imath} \in I_0$ be such that $\Phi_{\hat{\imath}}$ contradicts ρ_c, and specifically let $(s_{\mathbf{A}} \triangleleft \varepsilon \colon \rho) \preceq \Phi_{\hat{\imath}}$ be such that ρ contradicts ρ_c. If there is no such $\hat{\imath}$ then $I_0' = I_0$ and we're done.

As all sub-strategies of ρ_c are in $\tilde{\rho}$ and $\hat{\imath} \in I_0$, $\Phi_{\hat{\imath}}$ doesn't contradict any sub-strategy of ρ_c.

Let $\rho_c = \tilde{\pi}_c.\,(\mathfrak{l}|\hat{\rho}_c)$. Following Definition A.5.19 there is a sequence of steps $\tilde{\pi}.\,(\mathfrak{l}|\hat{\rho})$ contained in ρ 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 ρ (by hypothesis Γ is consistent) and ρ_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 Φ (first point in Definition A.5.40), $\bar{q}_{\mathbf{R}} \triangleleft \varepsilon_q \colon \rho_q.\,\phi_q \preceq \Phi_i$ where ρ_q is a precursor of $\hat{\rho}_c$. Moreover (second point in Definition A.5.40), there is a precursor ρ_0 of ρ where $(\mathfrak{l}|\bullet)$ replaces $(\mathfrak{l}|\hat{\rho})$ and such that $(s_{\mathbf{A}} \triangleleft \varepsilon_0 \colon \rho_0) \preceq \Phi_i$.

Applying Definition A.5.34, $\Phi_i \hookrightarrow \Phi_i \vee \Phi_i'$ where Φ_i' is obtained from Φ_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 ρ_q is a precursor of $\hat{\rho}_c$, $\Phi_i' \hookrightarrow \Phi_i' \vee \Phi_i''$ where Φ_i'' is obtained from Φ_i' by further replacing ρ_q by $\hat{\rho}_c$ in the rules above.

As Φ 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 Φ_i doesn't contradict $\tilde{\rho}$ and Φ_j was obtained from Φ_i by moving sub-strategies around, Φ_j doesn't contradict $\tilde{\rho}$ either so we have $j \in I_0$. By construction Φ_j doesn't contradict ρ_c, so $j \in I_0'$ and therefore I_0' can't be empty. □

We introduce a few notations used by the annotated type system rules.

Having $\mathrm{n}(p) = a$, "ξ_p" is ξ_{I} if $p = a$, and ξ_{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 $\pi.\,(s_{\mathbf{A}} \triangleleft \varepsilon \colon \rho') \stackrel{\text{def}}{=} s_{\mathbf{A}} \triangleleft \varepsilon \colon (\pi.\,\rho')$ and $\pi.\,\Gamma \stackrel{\text{def}}{=} \Gamma$ 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 $(x_i)_k \triangleleft \varepsilon$ (resp., $(\overline{x_i})_k \triangleleft \varepsilon$) by $(x_i)_k \triangleleft \varepsilon \colon (\bullet|l)\,[\,i\,]$ (resp., $(\overline{x_i})_k \triangleleft \varepsilon \colon (\bullet|l)[\,\bar{\imath}\,]$). Regarding sums, $\big(\sum_i x_i\big)_{\mathbf{A}} \triangleleft \varepsilon$ becomes $\big(\sum_i x_i\big)_{\mathbf{A}} \triangleleft \varepsilon \colon (\bullet|l)\,[\,\sum_i i\,]$

Definition A.5.42 (Annotated Type System)

$$\frac{\begin{array}{c}\forall i : \big(\mathsf{sub}(G_i) = \{p_i\},\quad (\Sigma_i; \Phi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash' G_i{}^{l_i}.P_i\big) \\ \Xi_{\mathrm{E}} \preceq \bigwedge_i \Xi_{\mathrm{E}i} \\ \big(\Xi_{\mathrm{E}} \text{ has concurrent environment } p_{i'}\big) \Rightarrow \varepsilon = \bot\end{array}}{\big(\bigwedge_i \Sigma_i; \big((\sum_i p_i)_{\mathbf{A}} \triangleleft \varepsilon : \sum_i l_i\big) \wedge \bigvee_i \Phi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}}\big) \vdash' \sum_i G_i{}^{l_i}.P_i} \text{ (R-Sum)}$$

$$\frac{\begin{array}{c}\Gamma \vdash' P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x} \\ (\#(G) = 1 \text{ and } m' = \star) \Rightarrow \varepsilon = \bot \\ \sigma = \langle \tilde{\sigma}; \xi_{\mathrm{I}}; \xi_{\mathrm{O}} \rangle\end{array}}{\begin{array}{rl} \Big(p : \sigma;\ \blacktriangleleft p^m \wedge \bar{p}^{m'}\Big) & \odot \\ \Big(; p_{\mathbf{A}}^{\#(G)} \triangleleft \varepsilon : l \blacktriangleleft\Big) & \odot \\ \mathbf{!}_{\text{if } \#(G) = \omega}\ (\boldsymbol{\nu}\mathrm{bn}(G))\Big(\ (l|\bullet).\Gamma \triangleleft \bar{p}_{\mathbf{A}} & \odot \\ \bar{\sigma}[\tilde{x}]_l \triangleleft \bar{p}_{\mathbf{AR}} & \odot \\ (; p_{\mathbf{R}} \triangleleft \sigma[\tilde{x}] : l.\,(\xi_p : \bullet) \blacktriangleleft\ ) & \Big) \vdash' G^l.P \end{array}} \text{ (R-Pre)}$$

Table 6: Annotated Rules

The Annotated Type System works like the one in Section 6.5 but constructs strategies for each dependency statement, using the rules from Table 6 (that only contains the rules that are different from the ones in Table 5).

The following lemma is shown by a trivial structural inductive proof, as the behaviour of operators with respect to dependencies was not modified:

Lemma A.5.43 (Type System Equivalence) *Let* $(\Gamma; P)$ *be a typed process such that* $\Gamma \vdash P$. *then there is an annotated typed process* $(\Gamma'; P')$ *such that* $\Gamma' \vdash' P'$ *and* $\mathrm{ran}(\Gamma'; P') = (\Gamma; P)$.

Given P and the typing $\Gamma \vdash P$, 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 (R-Pre) for that prefix in the derivation for $\Gamma \vdash P$.

Lemma A.5.44 (Annotated Type System Soundness) *Let* $(\Gamma; P)$ *be an annotated typed process such that* $\Gamma \vdash' P$. *Then* $(\Gamma; P)$ *is consistent and complete.*

Proof 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 A.5.38

*Completeness of (*R-Par*) strategies.* Assume both Γ_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 Γ_1 and Γ_2. As $\odot$ is a logical homomorphism, the local behavioural statement of $\Gamma_1 \odot \Gamma_2$ before the closure operator is applied is given by $\bigvee_{j \in J} \Phi_j \odot \bigvee_{k \in K} \Phi_k = \bigvee_{j \in J, k \in K} (\Phi_j \odot \Phi_k)$.

Let $\tilde{\rho}$ be a choice set. As both Γ_i are pre-complete there are $j \in J$ and $k \in K$ such that both Φ_j and Φ_k are pre-complete with respect to $\tilde{\rho}$.

We show that the three points in Definition A.5.40 are satisfied by $\Phi' = \Phi_j \odot \Phi_k$ with respect to $\tilde{\rho}$:

- As all activeness strategies in Φ' originate from Φ_j and Φ_k which are pre-complete (wrt. $\tilde{\rho}$) by hypothesis, strategies in Φ' 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 ρ' of ρ 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 ρ'_i that is runnable in one of the P_i

  As ρ 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 ρ'_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 ρ 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 ρ.

  If π'_j substitutes $p' = \mathsf{sub}_{P_1}(\pi'_{j+1} . \cdots . \mathfrak{l}_n)$ by p (i.e. $\mathsf{obj}_{P_1}(\mathfrak{l}_j)[q] = p'$ and $\mathsf{obj}_{P_2}(\rho'_j)[q] = p$ for some q) then set $\hat{\rho} = \rho'_j[q]$ which is, by hypothesis, P_2-runnable, so we're done.

  In all other cases, ρ'_j is not used to compute $\mathsf{sub}_P(\rho)$ and it is safe to replace ρ'_j by $\bullet$ to get $\hat{\rho}$.

  We now have a precursor ρ' of ρ that is P_i-runnable. So, as by hypothesis Φ_j (this is for $i = 1$, take Φ_k if $i = 2$) is pre-complete for P_i with respect to $\tilde{\rho}$ and therefore contains a statement $p_{\mathbf{R}} \triangleleft \varepsilon : \rho_0 . \phi$ where ρ_0 is a precursor of ρ' (and therefore of ρ as well). By definition of the $\odot$ operator on behavioural statements, that exact same statement $p_{\mathbf{R}} \triangleleft \varepsilon : \rho_0 . \phi$ is contained in Φ' as well, as required.

- Assume Φ' contains an annotated activeness statement $p_{\mathbf{A}} \triangleleft \varepsilon : \rho_2$ and let ρ_1 be a precursor of ρ_2. Then, applying $\odot$ backwards, one of Φ_i contains the same statement, and as it is pre-complete with respect to $\tilde{\rho}$, contains a statement $p_{\mathbf{A}} \triangleleft \varepsilon' : \rho_0$ for some precursor of ρ_0. Applying $\odot$ back, the same statement $p_{\mathbf{A}} \triangleleft \varepsilon' : \rho_0$ is contained in Φ', 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 A.5.41 $\Gamma_1 \odot \Gamma_2$ is complete.

*Consistency of (*R-Pre*) strategies.* The typed process produced by this rule contains strategies in four places. The "l" strategy of local activeness 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 $\bar{p}_{\mathbf{A}} \wedge \bar{p}_{\mathbf{AR}}$ which is equivalent to the declared $\bar{p}_{\mathbf{AR}}$. Finally, for the last factor (continuation $(l|\bullet) . \Gamma \triangleleft \bar{p}_{\mathbf{A}}$), consider a dependency statement $q_{\mathbf{A}} \triangleleft \varepsilon_0 : \rho$ in Γ. After prefixing and adding a dependency, it becomes $q_{\mathbf{A}} \triangleleft (\varepsilon_0 \wedge \bar{p}_{\mathbf{A}}) : ((l|\bullet) . \rho)$. $\bullet$-steps being always runnable, ρ's runnability is preserved. Then (taking $P' =$

$G^l.P$) $\mathsf{dep}_{P'}((l|\bullet).\,\rho) = \bar{p}_{\mathbf{A}} \wedge \mathsf{dep}_{P'}(\rho) = \bar{p}_{\mathbf{A}} \wedge \mathsf{dep}_P(\rho)$, so $\mathsf{dep}_P(\rho) \succeq \varepsilon_0$ (which holds as Γ is consistent) implies $\mathsf{dep}_{P'}((l|\bullet).\,\rho) \succeq (\varepsilon_0 \wedge \bar{p}_{\mathbf{A}})$ which is what we needed. As all components are consistent, by Lemma A.5.38, 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 A.5.41, 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 $(\Sigma_i; \Phi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash' {G_i}^{l_i}.P_i$ 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 ρ_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 $\tilde{\rho}$. □

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 $\Gamma_0 \vdash' P_0$. By induction, Γ_0 is consistent and complete for P_0. We show that $(\Gamma'; G^l.P_0)$, where $\#(G) = \omega$ and Γ' is obtained from Γ_0 following (R-PAR), is consistent and complete as well. Let Γ be the type under replication, i.e. the composition of continuation, remote parameters and responsiveness. Remember (Definition 4.5.3 on page 21) that $!\,\Gamma \stackrel{\text{def}}{=} \Gamma \odot \Gamma \odot \cdots \odot \Gamma$ with as many instances as there are $\vee$-terms in Γ's local component (multiplied by two to make sure all multiplicities are $\star$ but we aren't concerned about multiplicities here). By Γ'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 $P \mid (\boldsymbol{\nu}\tilde{z})\left(P_1 \,|\, P_2 \,|\, \ldots \,|\, P_m\right)$ 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 Γ.

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 Γ_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 Γ^{2n} is complete.

A.5.7 Overall Proof

We may now formulate the proof of the Soundness Proposition as a corollary of the previous lemma:

Proof of Proposition 7.3.4

Let $(\Gamma; P)$ be a (non-annotated) typed process such that $\Gamma \vdash P$.

Let an arbitrary transition sequence

$$(\Gamma; P) = (\Gamma_0; P_0) \xrightarrow{\tilde{\mu}_0}\searrow (\Gamma'_0; P'_0) \tag{112}$$

By Subject Reduction (Prop. 7.3.2), there is $\Gamma''_0 \preceq \Gamma'_0$ such that $\Gamma''_0 \vdash P'_0$.

By the Type System Equivalence there is an annotated typed process $(\hat{\Gamma}'_0; \hat{P}'_0)$ such that $\hat{\Gamma}'_0 \vdash' \hat{P}'_0$ and $\mathrm{ran}(\hat{\Gamma}'_0; \hat{P}'_0) = (\Gamma''_0; P'_0)$.

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 6.4.4 and f is constructed as given in the completeness and correctness 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 $(\Gamma_i; P_i) = \mathrm{ran}(\hat{\Gamma}_i; \hat{P}_i)$ and $(\Gamma''_i; P'_i) = \mathrm{ran}(\hat{\Gamma}'_i; \hat{P}'_i)$.

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^{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 (112) gives a sequence matching the requirements of Definition 6.4.4. As this works for an arbitrary sequence we get $\Gamma \models P$.

□