Sized Types with Usages for Parallel Complexity of Pi-Calculus Processes

Patrick Baillot, Alexis Ghyselen, Naoki Kobayashi

Sized Types with Usages for Parallel Complexity of Pi-Calculus Processes

Patrick Baillot and Alexis Ghyselen
Univ Lyon, CNRS, ENS de Lyon, Universite Claude-Bernard Lyon 1
LIP, F-69342, Lyon Cedex 07, France

Naoki Kobayashi
The University of Tokyo
Japan

***Abstract*—We address the problem of analysing the complexity of concurrent programs written in Pi-calculus. We are interested in parallel complexity, or span, understood as the execution time in a model with maximal parallelism. A type system for parallel complexity has been recently proposed by Baillot and Ghyselen but it is too imprecise for non-linear channels and cannot analyse some concurrent processes. Aiming for a more precise analysis, we design a type system which builds on the concepts of sized types and usages. The new variant of usages we define accounts for the various ways a channel is employed and relies on time annotations to track under which conditions processes can synchronize. We prove that a type derivation for a process provides an upper bound on its parallel complexity.**

I. Introduction

Static analysis of complexity is a classic topic of program analysis. Among the various approaches developed for this purpose, those based on type systems offer the advantages of compositionality and verifiability. Several such systems have been devised for functional programs, e.g. [1]–[6]. If a program can be assigned a type, then one can obtain from the type derivation a bound for its execution on any input. Some of those systems are also equipped with efficient type inference algorithms allowing typing to be automated.

However, this question has been for the moment far less explored in the setting of concurrent programming. Indeed, concurrent programs are harder to analyse and reasoning on their behaviour is a challenging task. The π-calculus is an expressive formalism in which these questions can be explored. In this setting, type systems have been intensively and successfully employed to guarantee properties such as termination (e.g. [7], [8]), deadlock-freedom and livelock-freedom (e.g. [9]–[11]): see [12] for a survey.

We wish to develop an analogous type-based approach for the analysis of time complexity of concurrent programs, using the framework of π-calculus. This requires first to clarify which notion of time we want to consider. Two classic notions of time cost for parallel systems are sequential time (*work*) and parallel time (*span*). The first one counts the cumulated total execution time of all processes while the second one takes into account simultaneous parallel execution, assuming an unlimited number of processors is available. Thus, the span or maximally parallelized execution time is an idealized notion, but it is standard in algorithms analysis and can also be used to obtain in practice concrete bounds for execution on a given number of processors. A second aspect we want to deal with is parametricity, in the sense that we want to be able to derive bounds depending on the size of some inputs, just as in the case of complexity analysis of sequential programs.

In [13], [14] two operational semantics and type systems have been proposed for analysing respectively the work and the span of π-calculus processes. However, as stressed by the authors even though the type system for span seems useful for analysing the complexity of some parallel programs, it fails to type-check some examples of common concurrent programs, like semaphores. It is based on a combination of sized types and input/output types, in order to account suitably for the behaviour of channels w.r.t. reception and emission.

In the present paper we design a type system for span which can deal with a much wider range of concurrent computation patterns including the semaphores. For that we take inspiration from the notion of *type usage*, which has been introduced and explored in [10], [11], initially to guarantee absence of deadlock during execution. Type usages are a generalization of input/output types, and describe how each channel is used for input and output. This description is given as a kind of CCS process. We formalize the type system with usages and prove its soundness. We also show through a number of examples (including semaphores) that our type system is often much more expressive than the type system of Baillot and Ghyselen [13].

Paper outline We introduce in Sect. II the π-calculus and the notion of parallel complexity we consider. Sect. III is devoted to the definition of types with usages. Then in Sect. IV we prove the main result of this paper, the complexity soundness, and provide some examples. Finally related work is discussed in Sect. V.

II. The Pi-calculus with Semantics for Span

In this work, we take the π-calculus as a model of parallelism and concurrent systems. The main points of π-calculus are that processes can be composed in parallel, communications between processes happen with the use of channels, and channel names can be created dynamically, so the topology can change at runtime.

A. Syntax and Standard Semantics for π-Calculus

We take as syntax a synchronous π-calculus, with a constructor `tick` that generates the time complexity. More details about π-calculus and variants of the syntax can be found

in [15]. The sets of *variables*, *expressions* and *processes* are defined by the following grammar.

$$v := x, y, z \mid a, b, c \qquad e := v \mid \mathtt{0} \mid \mathtt{s}(e)$$

$$P := 0 \mid (P \mid Q) \mid !a(\widetilde{v}).P \mid a(\widetilde{v}).P \mid \overline{a}\langle\widetilde{e}\rangle.P \mid (\nu a)P$$
$$\mid \mathtt{tick}.P \mid \mathtt{match}\ e\ \{\mathtt{case}\ \mathtt{0} \mapsto P; \mathtt{case}\ \mathtt{s}(x) \mapsto Q\}$$

We use x, y, z as meta-variables for integer variables, and a, b, c as those for channel names. The notation $\widetilde{v}$ stands for a sequence of variables $v_1, v_2, \ldots, v_k$. Similarly, $\widetilde{e}$ denotes a sequence of expressions. We work up to α-renaming, and we write $P[\widetilde{v} := \widetilde{e}]$ to denote the subtitution of $\widetilde{e}$ for the free variables $\widetilde{v}$ in P. For the sake of simplicity, we consider only integers as base types below, but the results can be generalized to other algebraic data-types such as lists or booleans.

Intuitively, $P \mid Q$ stands for the parallel composition of P and Q. The process $a(\widetilde{v}).P$ represents an input: it stands for the reception on the channel a of a tuple of values identified by the variables $\widetilde{v}$ in the continuation P. The process $!a(\widetilde{v}).P$ is a replicated version of $a(\widetilde{v}).P$, it behaves like an infinite number of $a(\widetilde{v}).P$ in parallel. The process $\overline{a}\langle\widetilde{e}\rangle.P$ represents an output: it sends a sequence of expressions on the channel a, and continues as P. A process $(\nu a)P$ dynamically creates a new channel name a and then proceeds as P. We also have standard pattern matching on data types, and finally, the `tick` constructor incurs a cost of one in complexity but has no semantic relevance. We consider that this constructor is the only source of time complexity in a program. It can represent different cost models and it is more general than counting the number of reduction steps in a standard setting. For example, by adding a `tick` after each input, we can count the number of communications in a process. By adding it after each replicated input on a channel a, we can count the number of calls to a. And if we want to count the number of reduction steps, we can add a `tick` after each input and pattern matching.

We now describe the standard semantics for this calculus. The first step is to define a congruence relation $\equiv$ on those processes. It is defined as the least congruence containing:

$$P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$$

$$(\nu a)(P \mid Q) \equiv (\nu a)P \mid Q \text{ (when } a \text{ is not free in } Q)$$

Note that the last rule can always be applied from right to left by α-renaming. Also, one can see that contrary to usual congruence relation for the π-calculus, we do not consider the rule for replication ($!P \equiv\ !P \mid P$) as it will be captured by the semantics, and α-conversion is not taken as an explicit rule in the congruence. By associativity, we will often write parallel composition for any number of processes and not only two. Another way to see this congruence relation is that, up to congruence, a process is entirely described by a set of channel names and a multiset of guarded processes. Formally, we give the following definition.

$$\frac{}{!a(\widetilde{v}).P \mid \overline{a}\langle\widetilde{e}\rangle.Q \to !a(\widetilde{v}).P \mid P[\widetilde{v} := \widetilde{e}] \mid Q}$$

$$\frac{}{a(\widetilde{v}).P \mid \overline{a}\langle\widetilde{e}\rangle.Q \to P[\widetilde{v} := \widetilde{e}] \mid Q}$$

$$\frac{}{\mathtt{match}\ \mathtt{0}\ \{\mathtt{case}\ \mathtt{0} \mapsto P; \mathtt{case}\ \mathtt{s}(x) \mapsto Q\} \to P}$$

$$\frac{}{\mathtt{match}\ \mathtt{s}(e)\ \{\mathtt{case}\ \mathtt{0} \mapsto P; \mathtt{case}\ \mathtt{s}(x) \mapsto Q\} \to Q[x := e]}$$

$$\frac{P \to Q}{P \mid R \to Q \mid R} \qquad \frac{P \to Q}{(\nu a)P \to (\nu a)Q}$$

$$\frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q}$$

Fig. 1. Standard Reduction Rules

Definition 1 (Guarded Processes and Canonical Form)**.** *A process G is* guarded *if it has one of the following shapes:*

$$G := !a(\widetilde{v}).P \mid a(\widetilde{v}).P \mid \overline{a}\langle\widetilde{e}\rangle.P \mid \mathtt{tick}.P$$
$$\mid \mathtt{match}\ e\ \{\mathtt{case}\ \mathtt{0} \mapsto P; \mathtt{case}\ \mathtt{s}(x) \mapsto Q\}$$

We say that a process is in canonical form *if it has the form $(\nu\widetilde{a})(G_1 \mid \cdots \mid G_n)$ with $G_1, \ldots, G_n$ guarded processes.*

For each process P, there is a process in canonical form congruent to P; thus, it is sufficient to consider only processes in canonical form. Moreover, this canonical form is unique up to the ordering of names and processes, and up to congruence inside guarded processes.

We now define the usual reduction relation for the π-calculus, that we denote $P \to Q$. It is defined by the rules given in Figure 1. Note that substitution should be well-defined in order to do some reduction steps: channel names must be substituted by other channel names and base type variables can be substituted by any expression except channel names. However, when we will consider typed processes, this will always yield well-defined substitutions.

For now, this relation cannot reduce a process of the form $\mathtt{tick}.P$. So, we need to introduce a reduction rule for `tick`. We will define parallel complexity (*span*) by taking an expansion of the standard reduction.

B. Parallel Complexity : The Span

The notion of complexity we are interested in is the parallel one. Before presenting the semantics, we present with some simple examples what kind of properties we want for this parallel complexity.

First, we want a parallel complexity that works as if we had an infinite number of processors. So, on the process $\mathtt{tick}.0 \mid \mathtt{tick}.0 \mid \mathtt{tick}.0 \mid \cdots \mid \mathtt{tick}.0$ we want the complexity to be 1, whatever the number of `tick` in parallel.

Moreover, a reduction step with a zero-cost complexity (in our setting, this should mean all reduction steps except when we reduce a `tick`) should not harm this maximal parallelism. For example $a().\mathtt{tick}.0 \mid \overline{a}\langle\rangle.0 \mid \mathtt{tick}.0$ should also have complexity one, because intuitively this synchronization between the input and the output can be done independently of

the `tick` on the right, and then the `tick` on the left can be reduced in parallel with the `tick` on the right.

Finally, adding a `tick` should not change the behaviour of a process. For instance, consider the process $\texttt{tick}.a().P_0 \mid a().\texttt{tick}.P_1 \mid \overline{a}\langle\rangle$, where a is not used in P_0 and P_1. This process should have the complexity $\max(1+\mathsf{C}_0, 1+\mathsf{C}_1)$, where C_i denotes the complexity of the process P_i. Indeed, there are two possible reductions, either we reduce the tick, and then we synchronize the left input with the output, and continue with P_0, or we first do the synchronization with the right input and the output, we then reduce the ticks and continue as P_1.

We use the definition of span from [13]. It consists in introducing a new construction for processes, $m : P$, where m is an integer. A process using this constructor will be called an *annotated process*. Intuitively, this annotated process has the meaning P *with* m *ticks before*. The congruence relation $\equiv$ is then enriched with the following relations:

$$0 : P \equiv P \qquad m : (P \mid Q) \equiv (m : P) \mid (m : Q)$$

$$m : (\nu a)P \equiv (\nu a)(m : P) \qquad m : (n : P) \equiv (m + n) : P$$

So, zero tick is equivalent to nothing and ticks can be distributed over parallel composition as expressed by the second relation. Name creation can be done before or after ticks without changing the semantics and finally ticks can be grouped together.

With this congruence relation and this new constructor, the canonical form presented in Definition 1 is given a new shape.

Definition 2 (Canonical Form for Annotated Processes)**.** *An annotated process is in canonical form if it has the shape:*

$$(\nu\widetilde{a})(n_1 : G_1 \mid \cdots \mid n_m : G_m)$$

with $G_1, \ldots, G_m$ *guarded processes.*

The rules for the reduction relation $\Rightarrow$ are given in Figure 2. This semantics works as the usual semantics for π-calculus, but when doing a synchronization, only the maximal annotation is kept, and ticks are memorized in the annotations.

Parallel complexity or span is then defined by:

Definition 3 (Parallel Complexity)**.** *Let* P *be an annotated process. Its* local complexity $\mathcal{C}_\ell(P)$ *is defined by:*

$$\mathcal{C}_\ell(n : P) = n + \mathcal{C}_\ell(P) \qquad \mathcal{C}_\ell(P \mid Q) = \max(\mathcal{C}_\ell(P), \mathcal{C}_\ell(Q))$$

$$\mathcal{C}_\ell((\nu a)P) = \mathcal{C}_\ell(P) \qquad \mathcal{C}_\ell(G) = 0 \text{ if } G \text{ is a guarded process}$$

Equivalently, $\mathcal{C}_\ell(P)$ *is the maximal integer that appears in the canonical form of* P*. Then, for an annotated process* P*, its* global parallel complexity *is given by*

$$max\{n \mid P \Rightarrow^* Q \wedge \mathcal{C}_\ell(Q) = n\}$$

where $\Rightarrow^*$ *is the reflexive and transitive closure of* $\Rightarrow$*.*

This parallel complexity satisfies the following lemma [13]:

Lemma 1 (Reduction and Local Complexity)**.** *Let* P, P' *be annotated processes such that* $P \Rightarrow P'$*. Then,* $\mathcal{C}_\ell(P') \geq \mathcal{C}_\ell(P)$

So, in order to bound the complexity of an annotated process, we need to reduce it with $\Rightarrow$, and then take the maximal local complexity over all normal forms. If there is none, the complexity is the limit of the local complexity on those infinite reductions. One can see that this semantics respects the conditions given above.

Example 1. *Let us write* $\overline{a}\langle\rangle$ *to represent the process* $\overline{a}\langle\rangle.0$*. Consider this process, a kind of simplified semaphore:*

$$P := \texttt{tick}.a().\texttt{tick}.\overline{a}\langle\rangle \mid \texttt{tick}.a().\texttt{tick}.\overline{a}\langle\rangle \mid \overline{a}\langle\rangle$$

By reducing P *we obtain:*

$$\begin{aligned} P &\Rightarrow^2 1 : (a().\texttt{tick}.\overline{a}\langle\rangle) \mid 1 : (a().\texttt{tick}.\overline{a}\langle\rangle) \mid 0 : \overline{a}\langle\rangle \\ &\Rightarrow 1 : (a().\texttt{tick}.\overline{a}\langle\rangle) \mid 1 : (\texttt{tick}.\overline{a}\langle\rangle) \\ &\Rightarrow 1 : (a().\texttt{tick}.\overline{a}\langle\rangle) \mid 2 : \overline{a}\langle\rangle \\ &\Rightarrow 2 : (\texttt{tick}.\overline{a}\langle\rangle) \Rightarrow 3 : \overline{a}\langle\rangle \end{aligned}$$

This shows the process has at least complexity 3*. As all the other possible choices we could have made in the reduction steps are similar, the process has exactly complexity* 3*.*

III. Types with Usages

In order to introduce the relevance of usages for complexity, let us look at another example similar to Example 1.

Example 2 (Motivating Example)**.** *We define*

$$P := a().\texttt{tick}.\overline{a}\langle\rangle$$

Then, the complexity of $P \mid P \mid P \mid \cdots \mid P \mid \overline{a}\langle\rangle$ *is equal to the number of* P *in parallel.*

The goal of our work is to design a type system for processes such that if $\Gamma \vdash Q \triangleleft K$ then K is a bound on the complexity of Q, such as in [13]. Let us look at the rule for parallel composition. If we take a rule of the shape:

$$\frac{\Gamma \vdash Q_1 \triangleleft K_1 \qquad \Gamma \vdash Q_2 \triangleleft K_2}{\Gamma \vdash Q_1 \mid Q_2 \triangleleft K}$$

The first thing to see is that if we want to take into account the case where Q_1 and Q_2 are totally independent, we should take $K = \max(K_1, K_2)$. But, with our previous example, given a typing $\Gamma \vdash P \triangleleft K$, then we would obtain $\Gamma \vdash P \mid \cdots \mid P \triangleleft K$ and so we have two processes with different complexity when composed with $\overline{a}\langle\rangle$ that are not distinguished by typing. In [13], P was not typable and so this problem did not occur, but it implied some limitations on the expressivity of their type system. A better alternative is to separate contexts.

$$\frac{\Gamma \vdash Q_1 \triangleleft K_1 \qquad \Delta \vdash Q_2 \triangleleft K_2}{f(\Gamma, \Delta) \vdash Q_1 \mid Q_2 \triangleleft \max(K_1, K_2)}$$

This could for example lead to linear type systems, or usages [12]. As we want a type system that can type examples such as P, usages seem adapted. Indeed, the concept of reliability, a central notion of usages, that allows types to adapt compositionally, is especially useful here, as we will see in Example 4. The type system we present uses an adaptation of usages combined with sized types [5], [6], [16], in order to handle recursion by replicated input.

$$\frac{}{(n : a(\widetilde{v}).P) \mid (m : \overline{a}\langle\widetilde{e}\rangle.Q) \Rightarrow \max(m,n) : (P[\widetilde{v} := \widetilde{e}] \mid Q)} \qquad \frac{}{\texttt{tick}.P \Rightarrow 1 : P}$$

$$\frac{}{(n :!a(\widetilde{v}).P) \mid (m : \overline{a}\langle\widetilde{e}\rangle.Q) \Rightarrow (n :!a(\widetilde{v}).P) \mid (\max(m,n) : (P[\widetilde{v} := \widetilde{e}] \mid Q))}$$

$$\frac{}{\texttt{match 0 }\{\texttt{case 0} \mapsto P; \texttt{case s}(x) \mapsto Q\} \Rightarrow P} \qquad \frac{}{\texttt{match s}(e)\ \{\texttt{case 0} \mapsto P; \texttt{case s}(x) \mapsto Q\} \Rightarrow Q[x := e]}$$

$$\frac{P \Rightarrow Q}{P \mid R \Rightarrow Q \mid R} \qquad \frac{P \Rightarrow Q}{(\nu a)P \Rightarrow (\nu a)Q} \qquad \frac{P \Rightarrow Q}{(n : P) \Rightarrow (n : Q)} \qquad \frac{P \equiv P' \qquad P' \Rightarrow Q' \qquad Q' \equiv Q}{P \Rightarrow Q}$$

Fig. 2. Reduction Rules for Annotated Processes

A. Indices

First, we use integer indices to keep track of the size of values in a process. The main idea of those types in a sequential setting is to control recursive calls by ensuring a decrease in the sizes. In our setting, those sizes are useful to control replicated inputs.

Definition 4. *We take $\mathcal{V}$ a countable set of index variables, usually denoted by i,j or k. The set of* indices*, representing integers in $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$, is given by the following grammar.*

$$I, J := I_{\mathbb{N}} \;\big|\; \infty \qquad I_{\mathbb{N}} := i \;\big|\; f(I_{\mathbb{N}}, \ldots, I_{\mathbb{N}})$$

where $i \in \mathcal{V}$. The symbol f is an element of a given set of function symbols containing for example integers constants as nullary operators, addition and multiplication. We also assume that we have the subtraction as a function symbol, with $n - m = 0$ when $m \geq n$. We consider that each function symbol f of arity $\texttt{ar}(f)$ comes with an interpretation $[\![f]\!] : \mathbb{N}^{\texttt{ar}(f)} \to \mathbb{N}$.

Given an index valuation $\rho : \mathcal{V} \to \mathbb{N}$, we extend the interpretation of function symbols to indices.

$$[\![\infty]\!]_\rho = \infty \qquad [\![i]\!]_\rho = \rho(i)$$
$$[\![f(I_{\mathbb{N}}, \ldots, J_{\mathbb{N}})]\!]_\rho = [\![f]\!]([\![I_{\mathbb{N}}]\!]_\rho, \ldots, [\![J_{\mathbb{N}}]\!]_\rho)$$

This interpretation takes values in $\mathbb{N}_\infty$. For an index I, we denote the substitution of the occurrences of i in I by $J_{\mathbb{N}}$ with $I\{J_{\mathbb{N}}/i\}$. Note that $\infty\{J_{\mathbb{N}}/i\} = \infty$.

Definition 5 (Constraints on Indices)**.** *Let $\varphi \subset \mathcal{V}$ be a finite set of index variables. A* constraint *C on φ is an expression with the shape $I \bowtie J$ where I and J are indices with free variables in φ and $\bowtie$ denotes a binary relation on $\mathbb{N}_\infty$. Usually, we take $\bowtie \in \{\leq, <, =, \neq\}$. A finite set of constraints is denoted Φ.*

For a finite set $\varphi \subset \mathcal{V}$, we say that a valuation $\rho : \varphi \to \mathbb{N}$ *satisfies* a constraint $I \bowtie J$ on φ, noted $\rho \vDash I \bowtie J$ when $[\![I]\!]_\rho \bowtie [\![J]\!]_\rho$ holds. Similarly, $\rho \vDash \Phi$ holds when $\rho \vDash C$ for all $C \in \Phi$. Likewise, we note $\varphi; \Phi \vDash C$ when for all valuations ρ on φ such that $\rho \vDash \Phi$ we have $\rho \vDash C$.

Definition 6 (Notations for Indices)**.** *In order to harmonize notation, we extend some operations on indices $I_{\mathbb{N}}, J_{\mathbb{N}}$ to indices I, J. We will use the following operations:*

$$\infty + J = I + \infty = \infty \qquad \max(\infty, J) = \max(I, \infty) = \infty$$
$$\min(\infty, J) = \min(J, \infty) = J \qquad \infty - 1 = \infty$$

B. Usages

We use *usages* to express the channel-wise behaviour of a process. Our notion of usages has been inspired by the usages introduced in type systems for deadlock-freedom [10], [11], [17], but differs from the original one in a significant manner. We define usages as a kind of CCS processes [18] on a single channel, where each action is annotated with two time intervals.

The set of usages, ranged over by U or V, is given by:

$$U, V ::= 0 \mid (U|V) \mid \texttt{In}^{A_o}_{J_c}.U \mid \texttt{Out}^{A_o}_{J_c}.U \mid !U \mid U + V$$
$$A_o, B_o ::= [I, J] \qquad J_c, I_c ::= J \mid [I, J]$$

Given a set of index variables φ and a set of constraints Φ, for an interval $[I, J]$, we always require that $\varphi; \Phi \vDash I \leq J$. For an interval $A_o = [I, J]$, we denote $\mathsf{Left}(A_o) = I$ and $\mathsf{Right}(A_o) = J$. In the original notion of usages [10], [11], [17], A_o and J_c were just numbers. The extension to intervals plays an important role in our analysis. Note that J_c is not always an interval, as it can be a single index J. However, this single index J should be understood as the interval $[-\infty, J]$.

Intuitively, a channel with usage 0 is not used at all. A channel of usage $U \mid V$ can be used according to U and V possibly in parallel. The usage $\texttt{In}^{A_o}_{J_c}.U$ describes a channel that may be used for input, and then used according to U. The two intervals A_o and J_c, called *obligation* and *capacity* respectively, are used to achieve a kind of assume-guarantee reasoning. The obligation A_o indicates a *guarantee* that if the channel is indeed used for input, then the input will become ready during the interval A_o. The capacity J_c indicates the *assumption* that if the environment performs a corresponding output, that output will be provided during the time interval J_c after the input becomes ready. For example, if a channel a has usage $\texttt{In}^{[1,1]}_{J_c}.0$, then the process $\texttt{tick}.a().0$ conforms to the usage, but $a().0$ and $\texttt{tick}.\texttt{tick}.a().0$ do not. Furthermore, if $J_c = [0, 1]$, and if a process $\texttt{tick}^k.\overline{a}$ is running in parallel with $\texttt{tick}.a().0$, then k belongs to the interval $[1, 1]+[0, 1] = [1, 2]$. Similarly, $\texttt{Out}^{A_o}_{J_c}.U$ has the same meaning but for output. The usage $!U$ denotes the usage U that can be replicated infinitely, and $U + V$ denotes a non-deterministic choice between the usages U and V. This is useful for example in a case of pattern matching where a channel can be used very differently in the

two branches. For the sake of conciseness, we may use $\alpha^{A_o}_{J_c}.U$ to denote either the usage $\mathtt{Out}^{A_o}_{J_c}.U$ or $\mathtt{In}^{A_o}_{J_c}.U$.

Recall that the obligation and capacity intervals in usages express a sort of assume-guarantee reasoning. We thus require that the assume-guarantee reasoning in a usage is "consistent" (or *reliable*, in the terminology of usages). For example, the usage $\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[1,1]}_{0}$ is reliable, since the part $\mathtt{In}^{[0,0]}_{[1,1]}$ assumes that a corresponding output will become ready at time 1, and the other part $\mathtt{Out}^{[1,1]}_{0}$ indeed guarantees that. Then, $\mathtt{Out}^{[1,1]}_{0}$ assumes that a corresponding input will be ready by the time the output becomes ready, and the part $\mathtt{In}^{[0,0]}_{[1,1]}$ guarantees that. In contrast, the usage $\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[2,2]}_{0}$ is problematic because, although the part $\mathtt{In}^{[0,0]}_{[1,1]}$ assumes that an output will be ready at time 1, $\mathtt{Out}^{[2,2]}_{0}$ provides the output only at time 2. The consistency on assume-guarantee reasoning must hold during the whole computation; for example, in the usage $\mathtt{In}^{[0,0]}_{[0,0]}.\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[0,0]}_{[0,0]}.\mathtt{Out}^{[2,2]}_{0}$, the assume/guarantee on the first input/output pair is fine, but the usage expressing the next communication: $\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[2,2]}_{0}$ is problematic. To properly define the reliability of usages during the whole computation, we first prepare a reduction semantics for usages, by viewing usages as CCS processes.

Definition 7 (Congruence for Usages)**.** *Congruence on usages is defined as the least congruence relation closed under:*

$$U \mid 0 \equiv U \qquad U \mid V \equiv V \mid U \qquad U \mid (V \mid W) \equiv (U \mid V) \mid W$$

$$!0 \equiv 0 \qquad !U \equiv !U \mid U \qquad !(U \mid V) \equiv !U \mid !V \qquad !!U \equiv !U$$

We have the usual relations for parallel composition. We also add a relation defining replication $!U \equiv !U \mid U$. The other relations allow manipulation of replication. This will be useful for the subusage relation, as it can increase the set of typable programs.

Now that we have congruence, as before, we give the reduction semantics. Let us first introduce some notations.

Definition 8 (Operations on Usages)**.** *We define the operations $\oplus$, $\sqcup$, and $+$ by:*

$$\begin{aligned}
&A_o \oplus J = [0, \mathsf{Left}(A_o) + J] \\
&A_o \oplus [I, J] = [\mathsf{Right}(A_o) + I, \mathsf{Left}(A_o) + J] \\
&[I, J] \sqcup [I', J'] = [\max(I, I'), \max(J, J')] \\
&[I, J] + [I', J'] = [I + I', J + J']
\end{aligned}$$

Note that $\oplus$ is an operation that takes an obligation interval and a capacity and returns an interval. This is where we see the fact that a capacity J alone should be understood as the interval $[-\infty, J]$. Indeed, when considering a single index, the lower bound of the sum becomes 0. So, in particular, we have $A_o \oplus J \neq A_o \oplus [0, J]$ in general. A case where we need this single index will be explained in Example 3.

The delaying operation $\uparrow^{A_o} U$ on usages is defined by:

$$\begin{aligned}
&\uparrow^{A_o} 0 = 0 \qquad \uparrow^{A_o} (U \mid V) = \uparrow^{A_o} U \mid \uparrow^{A_o} V \\
&\uparrow^{A_o} (U + V) = \uparrow^{A_o} U + \uparrow^{A_o} V \\
&\uparrow^{A_o} \alpha^{B_o}_{J_c}.U = \alpha^{A_o + B_o}_{J_c}.U \qquad \uparrow^{A_o} (!U) = !(\uparrow^{A_o} U)
\end{aligned}$$

We also define $[I, J] + J_c$ and thus $\uparrow^{J_c} U$ by extending the operation with: $[I, J] + J' = [I, J + J']$.

Intuitively, a usage $\uparrow^{A_o} U$ corresponds to the usage U delayed by a time approximated by the interval A_o. Given two obligations A_o and B_o, $A_o \sqcup B_o$ corresponds to an interval of time approximating the time for which those two obligations are respected. For example, if an input has the obligation to be ready in the interval of time $[4, 8]$ and an output has the obligation to be ready in the interval of time $[5, 7]$, then we know for sure that the input and the output will both be ready in the interval of time $[5, 8]$.

The reduction relation is given by the rules of Figure 3. The first rule means that to reduce a usage, we choose one input and one output, and then we trigger the communication between them. This communication occurs and does not lead to an error when the capacity of an action corresponds indeed to a bound on the time the dual action is defined. This is given by the relation $A_o \subseteq B_o \oplus J_c$. As an example, let us suppose that $B_o = [1, 3]$, and the time for which the output becomes ready is in fact 2, then the capacity J_c says that after two units of time, the synchronization should happen in the interval J_c. So, if we take $J_c = [5, 7]$ for example, then if we call t the time for which the dual input becomes ready, we must have $t \in [2 + 5, 2 + 7]$. This should be true for any time value in B_o, so we want that $\forall t' \in [1, 3], \forall t \in A_o, t \in [t' + 5, t' + 7]$, and this is equivalent to $A_o \subseteq B_o \oplus [5, 7] = [8, 8]$. Indeed, 8 is the only time that is in the three intervals $[6, 8]$, $[7, 9]$ and $[8, 10]$. The case where $J_c = J$ is a single index occurs when t can be smaller than t', and in this case we only ask that the upper bound is correct: $\forall t' \in B_o, \forall t \in A_o, t \leq t' + J$.

If the bound was incorrect, we trigger an error, see the second rule. In the case everything went well, the continuation is delayed by an approximation of the time when this communication occurs (see $\uparrow^{A_o \sqcup B_o}$ in the first rule). The idea is that we would like, after a communication, to synchronize the time of the continuation with the other subprocesses of a usage. As it is not easy to advance the time of all the other subprocesses, delaying the current subprocess leads to an easier semantics. In the rules for $U + V$, a reduction step in usages can also make a non-deterministic choice.

An error in a usage reduction means that the assume-guarantee reasoning was inconsistent. Keeping that in mind, we define what is a reliable usage, that is to say a usage with consistent time indications.

Definition 9 (Reliability)**.** *A usage U is* reliable *under $\varphi; \Phi$ when for any reduction from U using $\longrightarrow$, it does not lead to an error.*

Example 3. *Take the usage*

$$U := \mathtt{In}^{[1,1]}_{1}.\mathtt{Out}^{[1,1]}_{0} \mid \mathtt{In}^{[1,1]}_{1}.\mathtt{Out}^{[1,1]}_{0} \mid \mathtt{Out}^{[0,0]}_{[1,1]}$$

The only possible reduction step (with symmetry) is:

$$U \longrightarrow \mathtt{Out}^{[2,2]}_{0} \mid \mathtt{In}^{[1,1]}_{1}.\mathtt{Out}^{[1,1]}_{0}$$

as we have indeed $[1, 1] \subseteq [0, 0] \oplus [1, 1] = [1, 1]$ and $[0, 0] \subseteq [1, 1] \oplus 1 = [0, 2]$. Note that the capacity $[0, 1]$ instead of 1 for

$$\frac{\varphi;\Phi \vDash B_o \subseteq A_o \oplus I_c \qquad \varphi;\Phi \vDash A_o \subseteq B_o \oplus J_c}{\varphi;\Phi \vdash \mathtt{In}_{I_c}^{A_o}.U \mid \mathtt{Out}_{J_c}^{B_o}.V \longrightarrow \uparrow^{A_o \sqcup B_o}(U \mid V)}$$

$$\frac{\varphi;\Phi \nvDash (B_o \subseteq A_o \oplus I_c \wedge A_o \subseteq B_o \oplus J_c)}{\varphi;\Phi \vdash \mathtt{In}_{I_c}^{A_o}.U \mid \mathtt{Out}_{J_c}^{B_o}.V \longrightarrow \mathtt{err}}$$

$$\frac{}{\varphi;\Phi \vdash U + V \longrightarrow U} \qquad \frac{}{\varphi;\Phi \vdash U + V \longrightarrow V}$$

$$\frac{\varphi;\Phi \vdash U \longrightarrow U' \qquad U' \neq \mathtt{err}}{\varphi;\Phi \vdash U \mid V \longrightarrow U' \mid V}$$

$$\frac{\varphi;\Phi \vdash U \longrightarrow \mathtt{err}}{\varphi;\Phi \vdash U \mid V \longrightarrow \mathtt{err}}$$

$$\frac{U \equiv U' \qquad \varphi;\Phi \vdash U' \longrightarrow V' \qquad V' \equiv V}{\varphi;\Phi \vdash U \longrightarrow V}$$

Fig. 3. Reduction Rules for Usages

the input would not have worked since $[1,1] \oplus [0,1] = [1,2]$. *Then, we end the reduction with*

$$\mathtt{Out}_0^{[2,2]} \mid \mathtt{In}_1^{[1,1]}.\mathtt{Out}_0^{[1,1]} \longrightarrow \mathtt{Out}_0^{[3,3]}$$

since $[2,2] \subseteq [1,1] \oplus 1 = [0,2]$ *and* $[1,1] \subseteq [2,2] \oplus 0 = [0,2]$. *Thus, this usage is reliable. It corresponds for example to the usage of the channel* a *in the process* P *given in Example 1.*

$$\mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \overline{a}\langle\rangle$$

The obligation $[1,1]$ *corresponds to waiting exactly one tick. Then, the capacities say that once they are ready, the two input will indeed communicate before one time unit for any reduction. And at the end, we obtain an output available at time* 3*, and this output has no communication. One can see that those capacities and obligations give indeed the complexity of this process. Thus, we will ask in the type system that all usages are reliable, and so the time indications will give some complexity bounds on the behaviour of a channel.*

Example 4. *Let us take as another example a non-reliable usage. We take the previous example and add another input in parallel*

$$U := \mathtt{In}_1^{[1,1]}.\mathtt{Out}_0^{[1,1]} \mid \mathtt{In}_1^{[1,1]}.\mathtt{Out}_0^{[1,1]} \mid \mathtt{In}_1^{[1,1]}.\mathtt{Out}_0^{[1,1]} \mid \mathtt{Out}_{[1,1]}^{[0,0]}$$

Again, the reduction gives:

$$U \longrightarrow^* \mathtt{Out}_0^{[3,3]} \mid \mathtt{In}_1^{[1,1]}.\mathtt{Out}_0^{[1,1]} \longrightarrow \mathtt{err}$$

because $[1,1] \oplus 1 = [0,2]$*, so the capacity here is not a good assumption. Therefore, this usage is not reliable. However, if we change the usage to*

$$U := \mathtt{In}_2^{[1,1]}.\mathtt{Out}_0^{[1,1]} \mid \mathtt{In}_2^{[1,1]}.\mathtt{Out}_0^{[1,1]} \mid \mathtt{In}_2^{[1,1]}.\mathtt{Out}_0^{[1,1]} \mid \mathtt{Out}_{[1,1]}^{[0,0]}$$

this time we obtain a reliable channel. This example shows how reliability adapts compositionally.

We introduce another relation $U \sqsubseteq V$ called the *subusage* relation, which will be used later to define the subtyping relation. It is defined by the rules of Figure 4. The relation $U \sqsubseteq V$ intuitively means that any channel of usage U may also be used according to V. For example, $U \sqsubseteq 0$ says that we may not use a channel (usage equal to 0). Recall that an obligation and a capacity express a guarantee and an assumption respectively. The last but one rule says that it is safe to strengthen the guarantee and weaken the assumption. We use the relation $I_c \leq J_c$ to denote the relation $\subseteq$ on intervals, where a single index J is considered as the interval $[-\infty, J]$. The last rule can be understood as follows. The part $\uparrow^{A_o+J_c} V$ says that a channel may be used according to V only after the interval $A_o + J_c$. Since the action $\alpha_{J_c}^{A_o}$ is indeed finished during the interval $A_o + J_c$, we can move V to under the guard of $\alpha_{J_c}^{A_o}$. This last rule is especially useful for substitution, as explained in the example below.

Example 5. *Let us consider the process:*

$$P := a(r).r().b() \mid \overline{a}\langle b\rangle$$

Let us give usages to b *and* r*; here we omit time annotations for the sake of simplicity.*

$$U_r = \mathtt{In} \qquad U_b = \mathtt{In} \mid U_r$$

Indeed, r *is used only once as an input, and* b *is used as an input on the left, and it is sent to be used as* r *on the right. Thus, after a reduction step we obtain* $P \to b().b()$ *where* b *has usage* $U_b' = \mathtt{In}.\mathtt{In}$*. So, the channel* b *had usage* U_b *in* P*, but it ended up being used according to* U_b'*; that is valid since we have the subusage relation* $U_b \sqsubseteq U_b'$*.*

Before continuing to the type system, we give some intermediate results on subusages.

Lemma 2 (Properties of Subusage)**.** *For a set of index variables* φ *and a set of constraints* Φ *on* φ *we have:*

1) *If* $\varphi;\Phi \vdash U \sqsubseteq V$ *then for any interval* A_o*, we have* $\varphi;\Phi \vdash \uparrow^{A_o} U \sqsubseteq \uparrow^{A_o} V$*.*
2) *If* $\varphi;\Phi \vdash U \sqsubseteq V$ *and* $\varphi;\Phi \vdash V \longrightarrow V'$*, then there exists* U' *such that* $\varphi;\Phi \vdash U \longrightarrow^* U'$ *and* $\varphi;\Phi \vdash U' \sqsubseteq V'$ *(with* $\mathtt{err} \sqsubseteq U$ *for any usage* U*)*
3) *If* $\varphi;\Phi \vdash U \sqsubseteq V$ *and* U *is reliable under* $\varphi;\Phi$ *then* V *is reliable under* $\varphi;\Phi$*.*

The first point shows that subtyping is invariant by delaying. The second property means that the subusage relation serves as a simulation relation, and the last one means that the reliability is closed under the subusage relation. In our setting, it means that the subusage relation cannot lead to unsound complexity bounds. Some proof elements can be found in the Appendix A.

Finally, we also have the following lemmas, saying that delaying does not modify the behaviour of a type.

Lemma 3 (Invariance by Delaying)**.** *For any interval* A_o*:*

1) *If* $\varphi;\Phi \vdash \uparrow^{A_o} U \longrightarrow V'$ *then, there exists* V *such that* $\uparrow^{A_o} V = V'$ *and* $\varphi;\Phi \vdash U \longrightarrow V$*. (with* $\mathtt{err} = \uparrow^{A_o} \mathtt{err}$*)*
2) *If* $\varphi;\Phi \vdash U \longrightarrow V$ *then* $\varphi;\Phi \vdash \uparrow^{A_o} U \longrightarrow \uparrow^{A_o} V$*.*
3) U *is reliable under* $\varphi;\Phi$ *if and only if* $\uparrow^{A_o} U$ *is reliable under* $\varphi;\Phi$*.*

$$\frac{}{\varphi;\Phi\vdash U\sqsubseteq 0} \qquad \frac{i\in\{1;2\}}{\varphi;\Phi\vdash U_1+U_2\sqsubseteq U_i} \qquad \frac{\varphi;\Phi\vdash U\sqsubseteq U'}{\varphi;\Phi\vdash U+V\sqsubseteq U'+V} \qquad \frac{\varphi;\Phi\vdash V\sqsubseteq V'}{\varphi;\Phi\vdash U+V\sqsubseteq U+V'}$$

$$\frac{\varphi;\Phi\vdash U\sqsubseteq U'}{\varphi;\Phi\vdash U\mid V\sqsubseteq U'\mid V} \qquad \frac{\varphi;\Phi\vdash U\sqsubseteq U'}{\varphi;\Phi\vdash !U\sqsubseteq !U'} \qquad \frac{U\equiv U' \qquad \varphi;\Phi\vdash U'\sqsubseteq V' \qquad V\equiv V'}{\varphi;\Phi\vdash U\sqsubseteq V}$$

$$\frac{\varphi;\Phi\vdash U\sqsubseteq U' \qquad \varphi;\Phi\vdash U'\sqsubseteq U''}{\varphi;\Phi\vdash U\sqsubseteq U''} \qquad \frac{\varphi;\Phi\vdash U\sqsubseteq U'}{\varphi;\Phi\vdash \alpha^{A_o}_{J_c}.U\sqsubseteq \alpha^{A_o}_{J_c}.U'} \qquad \frac{\varphi;\Phi\vDash B_o\subseteq A_o \qquad \varphi;\Phi\vDash I_c\le J_c}{\varphi;\Phi\vdash \alpha^{A_o}_{I_c}.U\sqsubseteq \alpha^{B_o}_{J_c}.U}$$

$$\frac{}{\varphi;\Phi\vdash (\alpha^{A_o}_{J_c}.U)\mid(\uparrow^{A_o+J_c}V)\sqsubseteq \alpha^{A_o}_{J_c}.(U\mid V)}$$

Fig. 4. Subusage

In our setting, this lemma shows among other things that the `tick` constructor, or more generally the annotation $n : P$, does not break reliability.

C. Type System

Definition 10 (Usage Types)**.** *We define* types *by the following grammar:*

$$T,S ::= \mathsf{Nat}[I,J] \;\Big|\; ch(\widetilde{T})/U \;\Big|\; \forall\widetilde{i}.serv^K(\widetilde{T})/U$$

So, as expected, usual types for π-calculus are replaced by a type with usage, and we keep track of sizes for integers. An integer n of type $\mathsf{Nat}[I,J]$ must satisfy $I\le n\le J$.

Channels are classified into *server channels* (or just *servers*) and *simple channels*. All the inputs on a server channel must be replicated (as in $!a(\widetilde{v}).P$), while no input on a simple channel can be replicated. The type $\mathtt{ch}(\widetilde{T})/U$ describes a simple channel that is used for transmitting values of type $\widetilde{T}$ according to usage U. The type $\forall\widetilde{i}.\mathtt{serv}^K(\widetilde{T})/U$ describes a server channel that is used for transmitting values of type $\widetilde{T}$ according to usage U; the superscript K, which we call the *complexity* of a server, is an interval. It denotes the cost incurred when a server is invoked. Note that the server type allows polymorphism on index variables $\widetilde{i}$.

Also note that for a simple channel, only one type $\widetilde{T}$ is associated to all usages. So for example, in a channel of type $\mathtt{ch}(\mathsf{Nat}[I,J])/U$, at any time this channel is used, all messages must be integers between I and J.

The subtyping relation is defined by the rules of Figure 5. The only thing subtyping can do is to change the usage of a channel or modify the size bound on an integer.

In order to describe the type system for those types, we need to extend the previous operations on usages to partial operations on types and typing contexts with $\Gamma = v_1 : T_1,\dots,v_n : T_n$. The delaying of a type $\uparrow^{A_o}T$ is defined as the delaying of the usage for a channel or a server type, and it does nothing on integers. We also say that a type is reliable when it is an integer type, or when it is a server or channel type with a reliable usage. We define following operations:

Definition 11 (Parallel Composition of Types)**.** *The parallel composition of two types $T \mid T'$ is defined by:*

$$\mathsf{Nat}[I,J]\mid\mathsf{Nat}[I,J]=\mathsf{Nat}[I,J]$$

$$ch(\widetilde{T})/U\mid ch(\widetilde{T})/V = ch(\widetilde{T})/(U\mid V)$$

$$\forall\widetilde{i}.serv^K(\widetilde{T})/U\mid\forall\widetilde{i}.serv^K(\widetilde{T})/V=\forall\widetilde{i}.serv^K(\widetilde{T})/(U\mid V)$$

Definition 12 (Replication of Type)**.** *The replication of a type $!T$ is defined by:*

$$!\mathsf{Nat}[I,J]=\mathsf{Nat}[I,J] \qquad !ch(\widetilde{T})/U = ch(\widetilde{T})/(!U)$$

$$!\forall\widetilde{i}.serv^K(\widetilde{T})/U=\forall\widetilde{i}.serv^K(\widetilde{T})/(!U)$$

The (partial) operations on types defined above are extended pointwise to contexts. For example, for $\Gamma = v_1 : T_1,\dots,v_n : T_n$ and $\Delta = v_1 : T'_1,\dots,v_n : T'_n$, we define $\Gamma\mid\Delta = v_1 : T_1\mid T'_1,\dots,v_n : T_n\mid T'_n$. Note that this is defined just if Γ and Δ agree on the typing of integers and associate the same types (excluding usage) to names.

We also introduce the following notation.

Definition 13. *Given a capacity J_c and an interval $K = [K_1,K_2]$, we define $J_c;K$ by;*

$$J;[K_1,K_2]=[0,J+K_2]$$

$$[\infty,\infty];[K_1,K_2]=[0,0] \qquad [I_{\mathbb{N}},J];[K_1,K_2]=[0,J+K_2]$$

Intuitively, $J_c;K$ represents the complexity of an input/output process when the input/output has capacity J_c and the complexity of the continuation is K. $J_c=[\infty,\infty]$ means the input/output will never succeed (because there is no corresponding output/input); hence the complexity is 0. A case where this is useful is given later in Example 8. Otherwise, an upper-bound is given by $J+K_2$ (the time spent for the input/output to succeed, plus K). The lower-bound is 0, since the input/output may be blocked forever.

The type system is given in Figures 6 and 7. The typing rules for expressions are standard ones for sized types.

A type judgment is of the form $\varphi;\Phi;\Gamma\vdash P\triangleleft[I,J]$ where φ denotes the set of index variables, Φ is a set of constraints on index variables, and J is a bound on the parallel complexity of P under those constraints. This complexity bound J can also be seen as a bound on the open complexity of a process, that is to say the complexity of P in an environment corresponding to the types in Γ. For example, a channel with usage $\mathtt{In}_5^{[1,1]}$ alone cannot be reduced, as it is only used as an input. So, the typing $\cdot;\cdot;a:\mathtt{ch}()/\mathtt{In}_5^{[1,1]}\vdash \mathtt{tick}.a()\triangleleft[1,6]$ says that in an environment that may provide an output on the channel a within the time interval $[1,1]\oplus 5=[0,6]$, this process has a

$$\frac{\varphi;\Phi \vDash I' \leq I \qquad \varphi;\Phi \vDash J \leq J'}{\varphi;\Phi \vdash \mathsf{Nat}[I,J] \sqsubseteq \mathsf{Nat}[I',J']} \qquad \frac{\varphi;\Phi \vdash \widetilde{T} \sqsubseteq \widetilde{T'} \qquad \varphi;\Phi \vdash \widetilde{T'} \sqsubseteq \widetilde{T} \qquad \varphi;\Phi \vdash U \sqsubseteq V}{\varphi;\Phi \vdash \mathtt{ch}(\widetilde{T})/U \sqsubseteq \mathtt{ch}(\widetilde{T'})/V}$$

$$\frac{\varphi,\widetilde{i};\Phi \vdash \widetilde{T} \sqsubseteq \widetilde{T'} \qquad \varphi,\widetilde{i};\Phi \vdash \widetilde{T'} \sqsubseteq \widetilde{T} \qquad \varphi,\widetilde{i};\Phi \vDash K = K' \qquad \varphi;\Phi \vdash U \sqsubseteq V}{\varphi;\Phi \vdash \forall \widetilde{i}.\mathtt{serv}^{K}(\widetilde{T})/U \sqsubseteq \forall \widetilde{i}.\mathtt{serv}^{K'}(\widetilde{T'})/V}$$

Fig. 5. Subtyping Rules for Usage Types

$$\frac{v : T \in \Gamma}{\varphi;\Phi;\Gamma \vdash v : T} \qquad \frac{}{\varphi;\Phi;\Gamma \vdash \mathtt{0} : \mathsf{Nat}[0,0]}$$

$$\frac{\varphi;\Phi;\Gamma \vdash e : \mathsf{Nat}[I,J]}{\varphi;\Phi;\Gamma \vdash \mathtt{s}(e) : \mathsf{Nat}[I+1,J+1]}$$

$$\frac{\varphi;\Phi;\Delta \vdash e : T' \qquad \varphi;\Phi \vdash \Gamma \sqsubseteq \Delta \qquad \varphi;\Phi \vdash T' \sqsubseteq T}{\varphi;\Phi;\Gamma \vdash e : T}$$

Fig. 6. Typing Rules for Expressions

complexity bounded by 6. Similarly, the lower bound I is a lower bound on the parallel complexity of P. But in practice, this lower bound is often too imprecise.[1]

The (par) rule separates a context into two parts, and the complexity is the maximum over the two complexities, both for lower bound and upper bound. The (tick) rule shows the addition of a tick implies a delay of $[1,1]$ in the context and the complexity. The (nu) rules imposes that all names must have a reliable usage when they are created. In order to type a channel with the (ich) rule, the channel must have an input usage, with obligation $[0,0]$. Note that with the subusage relation, we have $\mathtt{In}^{A_o}_{J_c} \sqsubseteq \mathtt{In}^{[0,0]}_{J_c}$ if and only if $A_o = [0,I]$ for some I. So, this typing rule imposes that the lower-bound guarantee is correct, but the rule is not restrictive for upper-bound. This rule induces a delay of J_c in both context and complexity. Indeed, in practice this input does not happen immediately as we need to wait for output. This is where the assumption on when this output is ready, given by the capacity, is useful. The rule for output (och) is similar. For a server, the rule for input (iserv) is similar to (ich) in principles but differs in the way complexity is managed. Indeed, as a replicated input is never modified nor erased through a computation, giving it a non-zero complexity would harm the precision of the type system. Moreover, if this server represents for example a function on an integer with linear complexity, then the complexity of this server depends on the size of the integer it receives, that is why the complexity is transferred to the output rule on server, as one can see in the rule (oserv). Indeed, this rule (oserv) is again similar to (och) but the complexity of a call to the server is added in the rule. As we have polymorphism on servers, in order to type an output we need to find an instantiation on the indices $\widetilde{i}$, which is denoted by $\widetilde{I_{\mathbb{N}}}$ in this rule. Finally, the (case) rule is the only rule that modifies the set of constraints, and it gives information on the values the sizes can take. As explained in Example 9, those constraints are crucial in our sized type system. Note that contexts are not separated in this rule. In the typing for the expression this is not a problem since names are not useful for the typing of an integer. Then for both branches, it means that the usage of channels must be the same. However, because we have the choice usage ($U+V$), in practice we can use different usages in those two branches.

As an illustration of the type system, let us take back again the reliable usage described in Example 3 and show that it corresponds indeed to the typing of a in the process described in Example 1.

Example 6. *The typing derivation of the process in Example 1 is given in Figure 8. Note that the process*

$$\mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \overline{a}\langle\rangle$$

is also typable, in the same way, using the usage described in Example 4, and with complexity $[1,3]$. *However, we saw in Example 4 that the usage was not reliable, and that is why we do not obtain a valid complexity bound. If we take the reliable usage*

$$U := \mathit{In}_2^{[1,1]}.\mathit{Out}_0^{[1,1]} \mid \mathit{In}_2^{[1,1]}.\mathit{Out}_0^{[1,1]} \mid \mathit{In}_2^{[1,1]}.\mathit{Out}_0^{[1,1]} \mid \mathit{Out}_{[1,1]}^{[0,0]}$$

It gives us back the correct complexity bound $[1,4]$

So, our type system can adapt compositionally with the use of reliability. And we saw on this example that reliability is needed to obtain soundness. An example for the use of servers and sizes is given later, in Example 7.

Remark. *A careful reader may wonder why we need* intervals *for obligations and capacities, instead of single numbers. An informal justification is given in the Appendix C.*

IV. Soundness and Examples

The proof of soundness relies on standard lemmas for type systems, mainly substitution lemmas and subject reduction. Note that in order to work on the parallel reduction relation $\Rightarrow$, we need to consider annotated processes. So, in the following we will always consider P as an annotated process. We introduce the following typing rule, for the annotation:

$$\frac{\varphi;\Phi;\Gamma \vdash P \triangleleft K}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma \vdash m : P \triangleleft K + [m,m]}$$

This rule corresponds to a generalization of the rule for `tick`.

[1]This is because in the definition of $J_C;K$ in Definition 13, we pessimistically take into account the possibility that each input/output may be blokced forever. We can avoid the pessimistic estimation of the lower-bound by incorporating information about lock-freedom [17], [19].

$$\text{(zero)}\ \frac{}{\varphi;\Phi;\Gamma\vdash 0\triangleleft[0,0]} \qquad \text{(par)}\ \frac{\varphi;\Phi;\Gamma\vdash P\triangleleft K_1 \qquad \varphi;\Phi;\Delta\vdash Q\triangleleft K_2}{\varphi;\Phi;\Gamma\mid\Delta\vdash P\mid Q\triangleleft K_1\sqcup K_2} \qquad \text{(tick)}\ \frac{\varphi;\Phi;\Gamma\vdash P\triangleleft K}{\varphi;\Phi;\uparrow^{[1,1]}\Gamma\vdash \mathtt{tick}.P\triangleleft K+[1,1]}$$

$$\text{(ich)}\ \frac{\varphi;\Phi;\Gamma,a:\mathtt{ch}(\widetilde{T})/U,\widetilde{v}:\widetilde{T}\vdash P\triangleleft K}{\varphi;\Phi;\uparrow^{J_c}\Gamma,a:\mathtt{ch}(\widetilde{T})/\mathtt{In}^{[0,0]}_{J_c}.U\vdash a(\widetilde{v}).P\triangleleft J_c;K} \qquad \text{(iserv)}\ \frac{(\varphi,\widetilde{i});\Phi;\Gamma,a:\forall\widetilde{i}.\mathtt{serv}^K(\widetilde{T})/U,\widetilde{v}:\widetilde{T}\vdash P\triangleleft K}{\varphi;\Phi;\uparrow^{J_c}!\Gamma,a:\forall\widetilde{i}.\mathtt{serv}^K(\widetilde{T})/!\mathtt{In}^{[0,0]}_{J_c}.U\vdash !a(\widetilde{v}).P\triangleleft[0,0]}$$

$$\text{(och)}\ \frac{\varphi;\Phi;\Gamma',a:\mathtt{ch}(\widetilde{T})/V\vdash\widetilde{e}:\widetilde{T} \qquad \varphi;\Phi;\Gamma,a:\mathtt{ch}(\widetilde{T})/U\vdash P\triangleleft K}{\varphi;\Phi;\uparrow^{J_c}(\Gamma\mid\Gamma'),a:\mathtt{ch}(\widetilde{T})/\mathtt{Out}^{[0,0]}_{J_c}.(V\mid U)\vdash\overline{a}\langle\widetilde{e}\rangle.P\triangleleft J_c;K}$$

$$\text{(oserv)}\ \frac{\varphi;\Phi;\Gamma',a:\forall\widetilde{i}.\mathtt{serv}^K(\widetilde{T})/V\vdash\widetilde{e}:\widetilde{T}\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\} \qquad \varphi;\Phi;\Gamma,a:\forall\widetilde{i}.\mathtt{serv}^K(\widetilde{T})/U\vdash P\triangleleft K'}{\varphi;\Phi;\uparrow^{J_c}(\Gamma\mid\Gamma'),a:\forall\widetilde{i}.\mathtt{serv}^K(\widetilde{T})/\mathtt{Out}^{[0,0]}_{J_c}.(V\mid U)\vdash\overline{a}\langle\widetilde{e}\rangle.P\triangleleft J_c;(K'\sqcup K\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\})}$$

$$\text{(case)}\ \frac{\varphi;\Phi;\Gamma\vdash e:\mathsf{Nat}[I,J] \qquad \varphi;(\Phi,I\leq 0);\Gamma\vdash P\triangleleft K \qquad \varphi;(\Phi,J\geq 1);\Gamma,x:\mathsf{Nat}[I{-}1,J{-}1]\vdash Q\triangleleft K}{\varphi;\Phi;\Gamma\vdash \mathtt{match}\ e\ \{\mathtt{case}\ 0\mapsto P;\mathtt{case}\ \mathtt{s}(x)\mapsto Q\}\triangleleft K}$$

$$\text{(nu)}\ \frac{\varphi;\Phi;\Gamma,a:T\vdash P\triangleleft K \qquad T\text{ reliable}}{\varphi;\Phi;\Gamma\vdash(\nu a)P\triangleleft K} \qquad \text{(subtype)}\ \frac{\varphi;\Phi;\Delta\vdash P\triangleleft K \qquad \varphi;\Phi\vdash\Gamma\sqsubseteq\Delta \qquad \varphi;\Phi\vDash K\subseteq K'}{\varphi;\Phi;\Gamma\vdash P\triangleleft K'}$$

Fig. 7. Typing Rules for Processes with Usages

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot;\cdot;a:\mathtt{ch}()/\mathtt{Out}^{[0,0]}_0\vdash\overline{a}\langle\rangle\triangleleft[0,0]}}{\cdot;\cdot;a:\mathtt{ch}()/\mathtt{Out}^{[1,1]}_0\vdash\mathtt{tick}.\overline{a}\langle\rangle\triangleleft[1,1]}}{\cdot;\cdot;a:\mathtt{ch}()/(\mathtt{In}^{[0,0]}_1.\mathtt{Out}^{[1,1]}_0)\vdash a().\mathtt{tick}.\overline{a}\langle\rangle\triangleleft[0,2]}}{\cdot;\cdot;a:\mathtt{ch}()/(\mathtt{In}^{[1,1]}_1.\mathtt{Out}^{[1,1]}_0)\vdash\mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle\triangleleft[1,3]} \qquad \dfrac{}{\cdot;\cdot;a:\mathtt{ch}()/(\mathtt{Out}^{[0,0]}_{[1,1]})\vdash\overline{a}\langle\rangle\triangleleft[0,1]}}{\cdot;\cdot;a:\mathtt{ch}()/(\mathtt{In}^{[1,1]}_1.\mathtt{Out}^{[1,1]}_0\mid\mathtt{In}^{[1,1]}_1.\mathtt{Out}^{[1,1]}_0\mid\mathtt{Out}^{[0,0]}_{[1,1]})\vdash\mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle\mid\mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle\mid\overline{a}\langle\rangle\triangleleft[1,3]}$$

Fig. 8. Typing of Example 1

A. Intermediate Lemmas

We first give some usual and intermediate lemmas on the typing system.

Lemma 4 (Weakening)**.** *Let φ,φ' be disjoint set of index variables, Φ be a set of constraints on φ, Φ' be a set of constraints on (φ,φ'), Γ and Γ' be contexts on disjoint set of variables.*

1) *If $\varphi;\Phi;\Gamma\vdash e:T$ then $(\varphi,\varphi');(\Phi,\Phi');\Gamma,\Gamma'\vdash e:T$.*
2) *If $\varphi;\Phi;\Gamma\vdash P\triangleleft K$ then $(\varphi,\varphi');(\Phi,\Phi');\Gamma,\Gamma'\vdash P\triangleleft K$.*

We also show that we can remove some useless hypothesis.

Lemma 5 (Strengthening)**.** *Let φ be a set of index variables, Φ be a set of constraints on φ, and C a constraint on φ such that $\varphi;\Phi\vDash C$.*

1) *If $\varphi;(\Phi,C);\Gamma,\Gamma'\vdash e:T$ and the variables in Γ' are not free in e, then $\varphi;\Phi;\Gamma\vdash e:T$.*
2) *If $\varphi;(\Phi,C);\Gamma,\Gamma'\vdash P\triangleleft K$ and the variables in Γ' are not free in P, then $\varphi;\Phi;\Gamma\vdash P\triangleleft K$.*

Those two lemmas are proved easily by successive induction on the definitions in this paper. Then, we also have a lemma expressing that index variables can indeed be replaced by any index.

Lemma 6 (Index Substitution)**.** *Let φ be a set of index variables and $i\notin\varphi$. Let $J_{\mathbb{N}}$ be an index with free variables in φ. Then,*

1) *If $(\varphi,i);\Phi;\Gamma\vdash e:T$ then $\varphi;\Phi\{J_{\mathbb{N}}/i\};\Gamma\{J_{\mathbb{N}}/i\}\vdash e:T\{J_{\mathbb{N}}/i\}$.*
2) *If $(\varphi,i);\Phi;\Gamma\vdash P\triangleleft K$ then $\varphi;\Phi\{J_{\mathbb{N}}/i\};\Gamma\{J_{\mathbb{N}}/i\}\vdash P\triangleleft K\{J_{\mathbb{N}}/i\}$.*

We now present the variable substitution lemmas. In the setting of usages, this lemma is a bit more complex than usual. Indeed, we have a separation of contexts with the parallel composition, and we have to rely on subusage, especially the rule $\varphi;\Phi\vdash(\alpha^{A_o}_J.U)\mid(\uparrow^{A_o+J_c}V)\sqsubseteq\alpha^{A_o}_{J_c}.(U\mid V)$ as expressed in the Example 5 above. We put some emphasis on the following notation: when we write $\Gamma,v:T$ as a context in typing, it means that v does not appear in Γ.

Lemma 7 (Substitution)**.** *Let Γ and Δ be contexts such that $\Gamma\mid\Delta$ is defined. Then we have:*

1) *If $\varphi;\Phi;\Gamma,v:T\vdash e':T'$ and $\Delta\vdash e:T$ then $\varphi;\Phi;\Gamma\mid\Delta\vdash e'[v:=e]:T'$*
2) *If $\varphi;\Phi;\Gamma,v:T\vdash P\triangleleft K$ and $\Delta\vdash e:T$ then $\varphi;\Phi;\Gamma\mid\Delta\vdash P[v:=e]\triangleleft K$*

The first point is straightforward. It uses the fact that we have the relation $\varphi;\Phi\vdash U\sqsubseteq 0$ for any usage U, and so we

can use $\varphi; \Phi \vdash \Gamma \mid \Delta \sqsubseteq \Gamma$ in order to weaken Δ (similarly for Γ) if needed. The second point is more interesting. The easy case is when T is $\mathsf{Nat}[I, J]$ for some $[I, J]$. Then, we take a Δ that only uses the zero usage, and so $\Gamma \mid \Delta = \Gamma$ and everything becomes simpler. The more interesting cases are:

Lemma 8 (Difficult Cases of Substitution)**.** *We have:*

- *If* $\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{S})/W_0, c : \mathtt{ch}(\widetilde{S})/W_1 \vdash P \triangleleft K$ *then* $\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{S})/(W_0 \mid W_1) \vdash P[c := b] \triangleleft K$
- *If* $\varphi; \Phi; \Gamma, b : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{S})/W_0, c : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{S})/W_1 \vdash P \triangleleft K$ *then* $\varphi; \Phi; \Gamma, b : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{S})/(W_0 \mid W_1) \vdash P[c := b] \triangleleft K$

Again, the proof is done by induction on the typing derivation. Some proof elements are given in the Appendix D.

B. Subject Reduction and Soundness

We then explain the subject reduction. Let us first introduce a notation:

Definition 14 (Reduction for Contexts)**.** *We say that a context* Γ *reduces to a context* Γ' *under* $\varphi; \Phi$*, denoted* $\varphi; \Phi \vdash \Gamma \longrightarrow^* \Gamma'$ *when one of the following holds:*

- $\Gamma = \Gamma'$
- $\Gamma = \Delta, a : \mathtt{ch}(\widetilde{T})/U \qquad \varphi; \Phi \vdash U \longrightarrow^* U'$ $\Gamma' = \Delta, a : \mathtt{ch}(\widetilde{T})/U'$
- $\Gamma = \Delta, a : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/U \qquad \varphi; \Phi \vdash U \longrightarrow^* U'$ $\Gamma' = \Delta, a : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/U'$

So intuitively, Γ' is Γ after some reduction steps but only in a unique usage. Note that we obtain immediately that if all types in Γ are reliable then all types in Γ' are also reliable by definition of reliability.

We then formalize the subject reduction.

Theorem 1 (Subject Reduction)**.** *If* $\varphi; \Phi; \Gamma \vdash P \triangleleft K$ *with all types in* Γ *reliable and* $P \Rightarrow Q$ *then there exists* Γ' *with* $\varphi; \Phi \vdash \Gamma \longrightarrow^* \Gamma'$ *and* $\varphi; \Phi; \Gamma' \vdash Q \triangleleft K$*.*

In order to do that, we need first a lemma saying that the congruence relation behaves well with typing.

Lemma 9 (Congruence and Typing)**.** *Let* P *and* Q *be annotated processes such that* $P \equiv Q$*. Then,* $\varphi; \Phi; \Gamma \vdash P \triangleleft K$ *if and only if* $\varphi; \Phi; \Gamma \vdash Q \triangleleft K$*.*

This is proved by induction on $P \equiv Q$, and it relies on the congruence in usages and in the cases where we modify annotations, it mainly relies on the fact that delaying does not modify the behaviour of a type, as expressed by Lemma 3. Proof elements are given in the Appendix E.

And now that we can work up to the congruence relation with Lemma 9, Theorem 1 is proved by induction on $P \Rightarrow Q$. Without surprise, the most difficult case is for a communication, and it greatly relies on reliability, see Appendix F for details.

Finally, we conclude with the following theorem:

Theorem 2. *Let* P *be an annotated process and* n *be its global parallel complexity. Then, if* $\varphi; \Phi; \Gamma \vdash P \triangleleft [I, J]$ *with all types in* Γ *reliable, then we have* $\varphi; \Phi \vDash J \geq n$*. Moreover, if* Γ *does not contain any integers variables, we have* $\varphi; \Phi \vDash I \leq n$*.*

Proof. By Theorem 1, all reductions from P using $\Rightarrow$ conserve the typing. The context may be reduced too, but as reducibility does not harm reliability, we can still apply the subject reduction through all the reduction steps of $\Rightarrow$. Moreover, for a process Q, if we have a typing $\varphi; \Phi; \Gamma \vdash Q \triangleleft [I, J]$, then $J \geq \mathcal{C}_\ell(Q)$. Indeed, a constructor $n : P$ forces an increment of the complexity of n both in typing and in the definition of $\mathcal{C}_\ell(Q)$, and for parallel composition the typing imposes a complexity greater than the maximum as in the definition for $\mathcal{C}_\ell(Q)$. Thus, J is indeed a bound on the parallel complexity by definition. As for the lower bound, one can see that we do not always have $I \leq \mathcal{C}_\ell(Q)$ because of two guarded processes: the process $\mathtt{tick}.Q'$ and $\mathtt{match}\ e\ \{\mathtt{case}\ \mathtt{0} \mapsto Q_1; \mathtt{case}\ \mathtt{s}(x) \mapsto Q_2\}$. However, those two processes are not in normal form for $\Rightarrow$, because $\mathtt{tick}.Q' \Rightarrow 1 : Q'$ and as there are no integer variables in Γ, the pattern matching can also be reduced. Thus, from a process Q with possibly top guarded processes that are ticks or pattern matching, we can find Q' such that $Q \Rightarrow Q'$ and Q' has no guarded processes of this shape. And then, we obtain $I \leq \mathcal{C}_\ell(Q')$ which is smaller than the parallel complexity of Q by definition. □

C. Examples

Let us also present how sizes and polymorphism over indices in servers can type processes defined by replication such as the factorial. Please note that by taking inspiration from the typing in [13], using the type representation given in the Appendix B, more complicated examples of parallel programs such as the bitonic sort could be typed in our setting with a good complexity bound.

Example 7 (Factorial)**.** *Suppose given a function on expressions* $\mathtt{mult} : \mathsf{Nat}[I, J] \times \mathsf{Nat}[I', J'] \to \mathsf{Nat}[I * I', J * J']$*. In practice, this should be encoded as a server in* π*-calculus, but for the sake of simplicity we consider it as a function. We will describe the factorial and count the number of multiplications with* $\mathtt{tick}$*. For the sake of conciseness, we write* $\mathsf{Nat}[I]$ *to denotes* $\mathsf{Nat}[I, I]$*. We use the usual notation* $I!$ *to represent the factorial function in indices. The process representing factorial and its typing derivation are given in Figure 9. We denote* T *the following type:*

$$\forall i.\mathtt{serv}^{[0,i]}(\mathsf{Nat}[i], \mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]})/(!\mathtt{In}_\infty^{[0,0]}.\mathtt{Out}_0^{[0,\infty]})$$

This type is reliable and it would be reliable even if composed with any kind of output $\mathtt{Out}_0^{A_o}$ *if we want to call this server. Let us denote:*

$$T' = \forall i.\mathtt{serv}^{[0,i]}(\mathsf{Nat}[i], \mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]})/\mathtt{Out}_0^{[0,\infty]})$$

and we also pose:

$$S = \mathtt{ch}(\mathsf{Nat}[(i-1)!])/(\mathtt{Out}_0^{[i-1,i-1]} \mid \mathtt{In}_{[i-1,i-1]}^{[0,0]}) = S_1 \mid S_2$$

where S_1 *and* S_2 *are the expected separation of the usage. This type* S *is reliable under* $(i); (i \geq 1)$*. Thus, we give the typing described in Figure 9. From the type of* f*, we see on its complexity* $[0, i]$ *that it does at most a linear number of*

multiplications. Please note that the constraints that appear in a match *are useful since without them, we could not prove* $i; (i \leq 0) \vDash i! = i == 0$ *and* $i; (i \geq 1) \vDash i * (i\text{–}1)! = i!$. *Moreover, polymorphism over indices is necessary in order to find that the recursive call is made on a strictly smaller size* i–1.

Let us now justify the use of this operator $J_c; K$ in order to treat complexity.

Example 8 (Deadlock)**.** *Let us consider the process*

$$P := (\nu a)(\nu b)(a().\texttt{tick}.\overline{b}\langle\rangle \mid b().\texttt{tick}.\overline{a}\langle\rangle)$$

In order to understand the constraints we need to verify in order to type this process, we will give a typing with variables for obligation and capacity, and we will look at what values those variables can take. This is described in Figure 10.

First, as a *and* b *have exactly the same behaviour, they must have the same typing. And from just this, we already have some constraints to satisfy. Indeed, because of reliability, we have:*

$$B_o \subseteq A_o \oplus I_c \qquad A_o \subseteq B_o \oplus J_c$$

Moreover, in order to continue the typing, we must have $In_{I_c}^{A_o} \sqsubseteq In_{I'_c}^{[0,0]}$ *and* $Out_{J_c}^{B_o} \sqsubseteq \uparrow^{I'_c}\uparrow^{[1,1]} Out_{J'_c}^{[0,0]}$. *This gives us the additional constraints:*

$$[0,0] \subseteq A_o \qquad I_c \leq I'_c \qquad ([1,1] + I'_c) \subseteq B_o \qquad J_c \leq J'_c$$

So, if we put them together, we have:

$$([1,1] + I_c) \subseteq ([1,1] + I'_c) \subseteq B_o \subseteq A_o \oplus I_c$$

As $[0,0] \subseteq A_o$ *we have* $\mathsf{Left}(A_o) = 0$. *In order to have* $\mathsf{Right}([1,1] + I_c) \leq \mathsf{Right}(A_o \oplus I_c)$ *then we are forced to take* $I_c = \infty$ *or* $I_c = [I, \infty]$ *for some* I. *If we take such a capacity, this could induce a infinite upper bound. However, recall that we have the special case* $[\infty, \infty]; K = [0, 0]$. *So, if we can give an infinite lower bound to this capacity, we recover the complexity* 0 *of this deadlock. In fact, this is possible as described in Figure 11. Thus, the capacity* $[\infty, \infty]$, *describing input or output that will never be reduced, allows us to derive a complexity of zero.*

Example 9. *Finally, we describe informally an example for which our system can give a complexity, but fails to catch a precise bound. Let us consider the process:*

$$P := \texttt{tick}.!a(n).\texttt{match}\ n\ \{\texttt{case}\ 0 \mapsto 0; \texttt{case}\ \texttt{s}(m) \mapsto \overline{a}\langle m\rangle\}$$
$$\mid \overline{a}\langle 10\rangle \mid \texttt{tick}.\texttt{tick}.!a(n).0$$

This process has complexity 2. *However, if we want to give a usage to the server* a, *we must have a usage:*

$$!In_0^{[1,1]}.Out_1^{[0,0]} \mid Out_{[1,2]}^{[0,0]} \mid !In_0^{[2,2]}$$

We took as obligations the number of ticks before the action, and as capacity the minimal number for which we have reliability. So in particular, because of the capacity 1 *in the usage* $Out_1^{[0,0]}$, *typing the recursive call* $\overline{a}\langle m\rangle$ *increases the complexity by one, and so typing* n *recursive calls generates a complexity of* n *in the type system. So, in our setting, the complexity of this process can only be bounded by* 10. *Overall, this type system may not behave well when there are more than one replicated input process on each server channel, since an imprecision on a capacity for a recursive call leads to an overall imprecision depending on the number of recursive calls. This issue is the only source of imprecision we found with respect to the type system of [13]: see the conjecture in Section V.*

V. Related Work

Some contributions to the complexity analysis of parallel functional programs by means of type systems appear in [20], [21] but the languages studied do not offer the same communication primitives as the π-calculus and do not express concurrency.

Alternatively some other works address the problem of analysing the time complexity of distributed or concurrent systems [22]–[24]. They provide some interesting analysis on some instances of systems but are not targeted at a language with dynamic creation of processes and channel name passing as the π-calculus. Moreover, the techniques employed in [22], [23] based on analysis of flow graph or rely-guarantee reasoning do not seem to offer the same compositionality as type systems.

More recently [25], [26] proposed a type system with temporal session types to capture parallel cost models. They use several time modalities inspired by temporal logic, which give an interesting expressivity. However, as this work is in the setting of session types they do not consider in full generality the concurrent processes that can be written in π-calculus.

To our knowledge, the first work to study parallel complexity in π-calculus by types was given by Kobayashi [17], as another application of his usage type system for deadlock freedom. However, the definition of parallel complexity and thus the definition of usages and reliability in this work is quite different from ours, as its reduction does not take into account some non-deterministic paths and the extension with types that could capture recursive functions is suggested but not detailed nor formalized. Kobayashi [12], [19], [27] also used the notion of usages to reason about deadlocks, livelocks, and information flow, but he used a single number for each obligation and capacity (the latter is called a "capability" in his work); we have generalized the number to an interval to improve the precision of our analysis. More recently, Baillot and Ghyselen proposed in [13], [14] a type system with the same goal of analysing parallel complexity in π-calculus. Their type system builds on sized types and input/output types instead of usages. Because of that, they cannot manage successive uses of the same channel as in Example 1, as their names can essentially be used at only one specific time. In most cases, the time annotation used for channels in their setting corresponds to the sum of the lower bound for obligation and the upper bound for capacity in our setting. We conjecture the following result:

Conjecture (Comparison with [13])**.** *Suppose given a typing* $\varphi; \Phi; \Gamma_{i/o} \vdash_{i/o} P \triangleleft J$ *in the input/output sized type system of*

$$P := !f(n,r).\texttt{match}\ n\ \{\texttt{case}\ 0 \mapsto \overline{r}\langle 0\rangle; \texttt{case}\ \mathsf{s}(m) \mapsto (\nu r')(\overline{f}\langle m, r'\rangle \mid r'(x).\texttt{tick}.\overline{r}\langle mult(n,x)\rangle)\}$$

$$\dfrac{\dfrac{}{i;\cdot;n:\mathsf{Nat}[i] \vdash n:\mathsf{Nat}[i]} \quad \dfrac{\dfrac{}{i;(i\le 0)\vDash i! = i = 0}}{i;i\le 0;f:T',n:\mathsf{Nat}[i],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]} \vdash \overline{r}\langle 0\rangle \lhd [0,i]}\ \pi_1}{\dfrac{i;\cdot;f:T',n:\mathsf{Nat}[i],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]} \vdash \texttt{match}\ n\ \{\texttt{case}\ 0 \mapsto \overline{r}\langle 0\rangle; \texttt{case}\ \mathsf{s}(m) \mapsto (\nu r')(\overline{f}\langle m,r'\rangle \mid r'(x).\texttt{tick}.\overline{r}\langle mult(n,x)\rangle)\} \lhd [0,i]}{\cdot;\cdot;f:T \vdash !f(n,r).\texttt{match}\ n\ \{\texttt{case}\ 0 \mapsto \overline{r}\langle 0\rangle; \texttt{case}\ \mathsf{s}(m) \mapsto (\nu r')(\overline{f}\langle m,r'\rangle \mid r'(x).\texttt{tick}.\overline{r}\langle mult(n,x)\rangle)\} \lhd [0,0]}$$

with π_1 :

$$\dfrac{\dfrac{\cdots \vdash (m,r'):(\mathsf{Nat}[i],\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]})\{i\text{-}1/i\}}{i;i\ge 1;m:\mathsf{Nat}[i\text{-}1],r':S_1,f:T_1 \vdash \overline{f}\langle m,r'\rangle \lhd [0,i]} \quad \dfrac{\dfrac{\dfrac{\dfrac{(i;i\ge 1)\vDash i*(i\text{-}1)! = i!}{i;i\ge 1;n:\mathsf{Nat}[i],x:\mathsf{Nat}[(i\text{-}1)!] \vdash mult(n,x):\mathsf{Nat}[i!]}}{i;i\ge 1;n:\mathsf{Nat}[i],x:\mathsf{Nat}[(i\text{-}1)!],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[0,0]} \vdash \overline{r}\langle mult(n,x)\rangle \lhd [0,0]}}{i;i\ge 1;n:\mathsf{Nat}[i],x:\mathsf{Nat}[(i\text{-}1)!],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[1,1]} \vdash \texttt{tick}.\overline{r}\langle mult(n,x)\rangle \lhd [1,1]}}{i;i\ge 1;n:\mathsf{Nat}[i],r':S_2,r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]} \vdash r'(x).\texttt{tick}.\overline{r}\langle mult(n,x)\rangle \lhd [0,i]}}{i;i\ge 1;n:\mathsf{Nat}[i],m:\mathsf{Nat}[i\text{-}1],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]},f:T',r':S \vdash \overline{f}\langle m,r'\rangle \mid r'(x).\texttt{tick}.\overline{r}\langle mult(n,x)\rangle \lhd [0,i]}}{i;i\ge 1;n:\mathsf{Nat}[i],m:\mathsf{Nat}[i\text{-}1],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]},f:T' \vdash (\nu r')(\overline{f}\langle m,r'\rangle \mid r'(x).\texttt{tick}.\overline{r}\langle mult(n,x)\rangle) \lhd [0,i]}$$

Fig. 9. Representation and Typing of Factorial

$$\dfrac{\cdot;\cdot;a:\mathtt{ch}()/\mathtt{In}_{I_c}^{A_o},b:\mathtt{ch}()/\mathtt{Out}_{J_c}^{B_o} \vdash a().\texttt{tick}.\overline{b}\langle\rangle \lhd K_1 \qquad \cdot;\cdot;a:\mathtt{ch}()/\mathtt{Out}_{J_c}^{B_o},b:\mathtt{ch}()/\mathtt{In}_{I_c}^{A_o} \vdash b().\texttt{tick}.\overline{a}\langle\rangle \lhd K_2}{\dfrac{\cdot;\cdot;a:\mathtt{ch}()/(\mathtt{In}_{I_c}^{A_o} \mid \mathtt{Out}_{J_c}^{B_o}),b:\mathtt{ch}()/(\mathtt{Out}_{J_c}^{B_o} \mid \mathtt{In}_{I_c}^{A_o}) \vdash a().\texttt{tick}.\overline{b}\langle\rangle \mid b().\texttt{tick}.\overline{a}\langle\rangle \lhd K_1 \sqcup K_2}{\cdot;\cdot;\cdot \vdash (\nu a)(\nu b)(a().\texttt{tick}.\overline{b}\langle\rangle \mid b().\texttt{tick}.\overline{a}\langle\rangle) \lhd K_1 \sqcup K_2}}$$

Fig. 10. Typing Constraints for Example 8

$$\dfrac{\dfrac{\dfrac{}{\cdot;\cdot;a:\mathtt{ch}()/0,b:\mathtt{ch}()/\mathtt{Out}_0^{[0,0]} \vdash \overline{b}\langle\rangle \lhd [0,0]}}{\cdot;\cdot;a:\mathtt{ch}()/0,b:\mathtt{ch}()/\mathtt{Out}_0^{[1,1]} \vdash \texttt{tick}.\overline{b}\langle\rangle \lhd [1,1]} \quad \dfrac{\dfrac{}{\cdot;\cdot;a:\mathtt{ch}()/\mathtt{Out}_0^{[0,0]},b:\mathtt{ch}()/0 \vdash \overline{a}\langle\rangle \lhd [0,0]}}{\cdot;\cdot;a:\mathtt{ch}()/\mathtt{Out}_0^{[1,1]},b:\mathtt{ch}()/0 \vdash \texttt{tick}.\overline{a}\langle\rangle \lhd [1,1]}}{\dfrac{\cdot;\cdot;a:\mathtt{ch}()/\mathtt{In}_{[\infty,\infty]}^{[0,0]},b:\mathtt{ch}()/\mathtt{Out}_0^{[\infty,\infty]} \vdash a().\texttt{tick}.\overline{b}\langle\rangle \lhd [0,0]}{\cdot;\cdot;a:\mathtt{ch}()/(\mathtt{In}_{[\infty,\infty]}^{[0,0]} \mid \mathtt{Out}_0^{[\infty,\infty]}),b:\mathtt{ch}()/(\mathtt{Out}_0^{[\infty,\infty]} \mid \mathtt{In}_{[0,0]}^{[0,0]}) \vdash a().\texttt{tick}.\overline{b}\langle\rangle \mid b().\texttt{tick}.\overline{a}\langle\rangle \lhd [0,0]} \quad \cdot;\cdot;a:\mathtt{ch}()/\mathtt{Out}_0^{[\infty,\infty]},b:\mathtt{ch}()/\mathtt{In}_{[\infty,\infty]}^{[0,0]} \vdash b().\texttt{tick}.\overline{a}\langle\rangle \lhd [0,0]}}{\cdot;\cdot;\cdot \vdash (\nu a)(\nu b)(a().\texttt{tick}.\overline{b}\langle\rangle \mid b().\texttt{tick}.\overline{a}\langle\rangle) \lhd [0,0]}$$

Fig. 11. Typing of Example 8

[13], such that this process P has a linear use of channels. Then, there exists a reliable context Γ such that $\varphi;\Phi;\Gamma \vdash P \lhd [0,J]$.

More details and some intuitions are given in the Appendix B. So, on a simple use of names our system is strictly more precise if this conjecture is true. However, on other cases, like in Example 9, their system is more precise as the loss of precision because of usage does not happen in their setting. On the contrary, our setting has fairly more precision for processes with a non-trivial use of channels, as in Example 1. Indeed, their type system cannot express such a sequential use of channels, and so they cannot give a bound.

Some works have also been carried out in implicit computational complexity to characterize some complexity classes with process calculi [28]–[30] but for some languages which are not as expressive as the π-calculus and considering generally the work rather than the span. The paper [31] by contrast considers the π-calculus and causal (parallel) complexity, but the goal here is more to delineate a characterization of polynomial complexity rather than to give a sharp bound for a given process.

VI. Conclusion

We presented a type system built on sized types and usages such that a type derivation for a process gives an upper bound on the parallel complexity of this process. The type system relies on intervals in order to give an approximation of the sizes of integers in the process, and an approximation of the time an input or an output need to synchronize. In comparison to [13], we showed with examples that our type system can type some concurrent behaviour that was not captured in their type system, and on some subset of processes, we conjecture that we are strictly more precise.

Building on previous work by Kobayashi on type inference for usages [11], [19], we plan to investigate type inference, by the way of constraints solving procedures for indices.

REFERENCES

[1] J. Hoffmann, K. Aehlig, and M. Hofmann, "Resource aware ML," in *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, ser. Lecture Notes in Computer Science, vol. 7358. Springer, 2012, pp. 781–786.

[2] M. Hofmann and S. Jost, "Static prediction of heap space usage for first-order functional programs," in *Conference Record of POPL 2003: The 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New Orleans, Louisisana, USA, January 15-17, 2003*. ACM, 2003, pp. 185–197.

[3] J. Hoffmann and M. Hofmann, "Amortized resource analysis with polynomial potential," in *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, ser. Lecture Notes in Computer Science, vol. 6012. Springer, 2010, pp. 287–306.

[4] J. Hoffmann, K. Aehlig, and M. Hofmann, "Multivariate amortized resource analysis," *ACM Trans. Program. Lang. Syst.*, vol. 34, no. 3, pp. 14:1–14:62, 2012.

[5] U. Dal Lago and M. Gaboardi, "Linear dependent types and relative completeness," in *Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on*. IEEE, 2011, pp. 133–142.

[6] M. Avanzini and U. Dal Lago, "Automating sized-type inference for complexity analysis," *Proceedings of the ACM on Programming Languages*, vol. 1, no. ICFP, p. 43, 2017.

[7] Y. Deng and D. Sangiorgi, "Ensuring termination by typability," *Information and Computation*, vol. 204, no. 7, pp. 1045 – 1082, 2006.

[8] R. Demangeon, D. Hirschkoff, N. Kobayashi, and D. Sangiorgi, "On the complexity of termination inference for processes," in *Trustworthy Global Computing, Third Symposium, TGC 2007, Sophia-Antipolis, France, November 5-6, 2007, Revised Selected Papers*, ser. Lecture Notes in Computer Science, G. Barthe and C. Fournet, Eds., vol. 4912. Springer, 2007, pp. 140–155. [Online]. Available: https://doi.org/10.1007/978-3-540-78663-4_11

[9] N. Kobayashi, "A partially deadlock-free typed process calculus," in *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*. IEEE Computer Society, 1997, pp. 128–139. [Online]. Available: https://doi.org/10.1109/LICS.1997.614941

[10] E. Sumii and N. Kobayashi, "A generalized deadlock-free process calculus," in *Proc. of Workshop on High-Level Concurrent Language (HLCL'98)*, ser. ENTCS, vol. 16(3), no. 3, 1998, pp. 55–77.

[11] N. Kobayashi, S. Saito, and E. Sumii, "An implicitly-typed deadlock-free process calculus," in *CONCUR 2000 — Concurrency Theory*, C. Palamidessi, Ed. Springer Berlin Heidelberg, 2000, pp. 489–504.

[12] N. Kobayashi, "Type systems for concurrent programs," in *Formal Methods at the Crossroads. From Panacea to Foundational Support*. Springer, 2003, pp. 439–453.

[13] P. Baillot and A. Ghyselen, "Types for complexity of parallel computation in pi-calculus," in *To appear on 30th European Symposium on Programming, ESOP 2021*, 2021.

[14] ——, "Types for Complexity of Parallel Computation in Pi-calculus (Technical Report)," Oct. 2020, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02961427

[15] D. Sangiorgi and D. Walker, *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press, 2003.

[16] J. Hughes, L. Pareto, and A. Sabry, "Proving the correctness of reactive systems using sized types," in *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1996, pp. 410–423.

[17] N. Kobayashi, "A type system for lock-free processes," *Information and Computation*, vol. 177, no. 2, pp. 122 – 159, 2002.

[18] R. Milner, *Communication and concurrency*, ser. PHI Series in computer science. Prentice Hall, 1989.

[19] N. Kobayashi, "Type-based information flow analysis for the π-calculus," *Acta Informatica*, vol. 42, no. 4-5, pp. 291–347, 2005.

[20] J. Hoffmann and Z. Shao, "Automatic static cost analysis for parallel programs," in *Programming Languages and Systems*, J. Vitek, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 132–157.

[21] S. Gimenez and G. Moser, "The complexity of interaction," in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, 2016, pp. 243–255.

[22] E. Albert, J. Correas, E. B. Johnsen, and G. Román-Díez, "Parallel cost analysis of distributed systems," in *Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings*, ser. Lecture Notes in Computer Science, vol. 9291. Springer, 2015, pp. 275–292.

[23] E. Albert, A. Flores-Montoya, S. Genaim, and E. Martin-Martin, "Rely-guarantee termination and cost analyses of loops with concurrent interleavings," *Journal of Automated Reasoning*, vol. 59, no. 1, pp. 47–85, 2017.

[24] E. Giachino, E. B. Johnsen, C. Laneve, and K. I. Pun, "Time complexity of concurrent programs - - A technique based on behavioural types -," in *Formal Aspects of Component Software - 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 9539. Springer, 2016, pp. 199–216.

[25] A. Das, J. Hoffmann, and F. Pfenning, "Parallel complexity analysis with temporal session types," *Proc. ACM Program. Lang.*, vol. 2, no. ICFP, pp. 91:1–91:30, 2018.

[26] ——, "Work analysis with resource-aware session types," in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. ACM, 2018, pp. 305–314.

[27] N. Kobayashi, "A new type system for deadlock-free processes," in *International Conference on Concurrency Theory*. Springer, 2006, pp. 233–247.

[28] A. Madet and R. M. Amadio, "An elementary affine λ-calculus with multithreading and side effects," in *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, ser. Lecture Notes in Computer Science, vol. 6690. Springer, 2011, pp. 138–152.

[29] P. Di Giamberardino and U. Dal Lago, "On session types and polynomial time," *Mathematical Structures in Computer Science*, vol. -1, 2015.

[30] U. Dal Lago, S. Martini, and D. Sangiorgi, "Light logics and higher-order processes," *Mathematical Structures in Computer Science*, vol. 26, no. 6, pp. 969–992, 2016.

[31] R. Demangeon and N. Yoshida, "Causal computational complexity of distributed processes," in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS '18. ACM, 2018, pp. 344–353.

APPENDIX

In the appendix, we describe some proofs elements of lemmas and theorems given in the paper.

A. Proprieties of Subusage

First, we have the following lemma:

Lemma 10. *If* $\varphi;\Phi \vDash B_o \subseteq A_o$ *then* $\varphi;\Phi \vdash (\uparrow^{A_o} U) \sqsubseteq (\uparrow^{B_o} U)$

This can be proved easily by induction on U. We now give elements of proof for Point 2 and Point 3 of Lemma 2. For Point 2, we can see two possible directions, either proceed by induction on $U \sqsubseteq V$ and then do a case analysis on $V \longrightarrow V'$ on proceed by induction on $V \longrightarrow V'$ and then do a case analysis on $U \sqsubseteq V$. In both cases, the definition of subusage with the transitivity rule and congruence that can be used everywhere make the proof complicated. So, we chose to first simplify the definition of subusage.

Definition 15 (Decomposition of Subusage)**.** *Let us call* $\sqsubseteq_{ct}$ *the relation defined by the rule of Figure 4 without using congruence and transitivity. Then, we define* $\sqsubseteq_t$ *as:*

$$\frac{U \equiv U' \qquad \varphi;\Phi \vdash U' \sqsubseteq_{ct} V' \qquad V' \equiv V'}{\varphi;\Phi \vdash U \sqsubseteq_t V}$$

And we have the following lemma.

Lemma 11 (Decomposition of Subusage)**.** *The subusage relation* $\sqsubseteq$ *is equivalent to the reflexive and transitive closure of* $\sqsubseteq_t$*.*

The proof can be done by induction on $\sqsubseteq$, and it relies mainly on the fact that congruence can be used in any context, so we can use it at the end. In the same way, as the subusage relation can also be used in any context, the transitivity can be done at the end. So, with this lemma we got rid of the complicated rules by putting them always at the end of a derivation. Moreover, note that it is easy to describe exhaustively subtyping for $\sqsubseteq_t$. Let us drop the $\varphi;\Phi$ notation for the sake of conciseness, and we have:

Lemma 12 (Exhaustive Description of Subusage)**.** *Let us use* $* \in \{\cdot, !\}$*. Then,* $*U$ *denotes either* U *or* $!U$ *according to the value of* $*$*. If* $U \sqsubseteq_t V$*, then one of the following cases hold.*

$U \equiv (U_0 \mid *U_1) \qquad V \equiv U_0$

$U \equiv U_0 \mid *(U_1 + U_2) \qquad V \equiv U_0 \mid *U_i \qquad i \in \{1;2\}$

$U \equiv U_0 \mid *\alpha^{A_o}_{J_c}.W \qquad V \equiv U_0 \mid *\alpha^{A_o}_{J_c}.W' \qquad W \sqsubseteq_{ct} W'$

$U \equiv U_0 \mid *(U_1 + U_2) \qquad V \equiv U_0 \mid *(U_1' + U_2) \qquad U_1 \sqsubseteq_{ct} U_1'$

$U \equiv U_0 \mid *(U_1 + U_2) \qquad V \equiv U_0 \mid *(U_1 + U_2') \qquad U_2 \sqsubseteq_{ct} U_2'$

$U \equiv U_0 \mid *\alpha^{A_o}_{J_c}.W \qquad V \equiv U_0 \mid *\alpha^{A_o'}_{J_c'}.W \qquad A_o' \subseteq A_o \qquad J_c \leq J_c'$

$U \equiv U_0 \mid *((\alpha^{A_o}_{J_c}.W) \mid (\uparrow^{A_o+J_c} U_1)) \qquad V \equiv U_0 \mid *\alpha^{A_o}_{J_c}.(W \mid U_1)$

This proof is done directly by induction on $U \sqsubseteq_{ct} V$. The replication context rules works because we have $!(U_0 \mid W) \equiv !U_0 \mid !W$ and $!!W \equiv !W$. We now go back to Point 2 of Lemma 2.

Proof. We rely on Lemma 11. So, we will prove first this intermediate lemma:

Lemma 13. *If* $U \sqsubseteq_t V$ *and* $V \to V'$*, then there exists* U' *such that* $U \to^* U'$ *and* $U' \sqsubseteq V'$

Then, if this lemma is proved, we conclude by transitivity of the propriety. So, we only have to prove We will also use the following lemma

Lemma 14. *If* $V \equiv U_0 \mid !U_1$ *and* $V \longrightarrow V'$ *then* $V' \equiv W \mid !U_1$ *with* $U_0 \mid U_1 \longrightarrow W$

Indeed, there are three cases for $V \longrightarrow V'$. Either it is only a reduction step in U_0 independently of $!U_1$, either it is a reduction step within U_1 (note that one copy is always sufficient), either it is a synchronization between one action in U_0 and one action in U_1. In the first case, the lemma is true because we can arbitrarily add U_1. In the same way, in the second case the lemma is correct because we can just ignore U_0. In the third case, the lemma is verified since we allow this synchronization between U_0 and U_1.

We now start the proof. We proceed by induction on $U \sqsubseteq_{ct} V$, and we use the exhaustive description given by Lemma 12. We always consider the case $* =!$ as it is the harder of the two cases. Let us give some interesting cases:

- $$U \equiv U_0 \mid !\alpha^{A_o}_{J_c}.W \qquad V \equiv U_0 \mid !\alpha^{A_o}_{J_c}.W' \qquad W \sqsubseteq_{ct} W'$$

  The easy case is when V' is obtained by a reduction step in U_0. So, we suppose that the reduction step is a synchronization between U_0 and $\alpha^{A_o}_{J_c}.W'$. So, we have:

  $$U_0 \equiv U_0' \mid \overline{\alpha}^{B_o}_{I_c}.W_0$$

  If $V' = \texttt{err}$, then we take $U' = \texttt{err}$ and concludes this case. Otherwise, we have:

  $$V' \equiv U_0' \mid !\alpha^{A_o}_{J_c}.W' \mid \uparrow^{(A_o \sqcup B_o)}(W_0 \mid W')$$

  So, we take

  $$U' = U_0' \mid !\alpha^{A_o}_{J_c}.W \mid \uparrow^{A_o \sqcup B_o}(W_0 \mid W)$$

  And, we have indeed:

  $$U \longrightarrow U' \qquad U' \sqsubseteq V'$$

  The fact that $U' \sqsubseteq V'$ is given by the previous point of Lemma 2.

- $$U \equiv U_0 \mid !\alpha^{A_o}_{J_c}.W \qquad V \equiv U_0 \mid !\alpha^{A_o'}_{J_c'}.W$$

  $$A_o' \subseteq A_o \qquad J_c \leq J_c'$$

  Again, the only interesting case is when the synchronization is not only in U_0. So, we have:

  $$U_0 \equiv U_0' \mid !\overline{\alpha}^{B_o}_{I_c}.W_0$$

  If we have $A_o \subseteq B_o \oplus I_c$ and $B_o \subseteq A_o \oplus J_c$ then

  $$V' \equiv U_0' \mid !\alpha^{A_o'}_{J_c'}.W \mid \uparrow^{A_o' \sqcup B_o}(W_0 \mid W)$$

because $A'_o \subseteq A_o$ and $J_c \leq J'_c$ so we have $A'_o \subseteq B_o \oplus I_c$ and $B_o \subseteq A_o \oplus J'_c$. Thus, we take

$$U' = U'_0 \,|\, !\alpha^{A_o}_{J_c}.W \mid \uparrow^{A_o \sqcup B_o}(W_0 \mid W)$$

We have indeed $U \longrightarrow U'$ and $U' \sqsubseteq V'$ by Lemma 10. Otherwise, we obtain an error for U' and so it indeed is a subusage of V'.

- $$U \equiv U_0 \,|\, !((\alpha^{A_o}_{J_c}.W) \mid (\uparrow^{A_o+J_c} U_1))$$
$$V \equiv U_0 \,|\, !\alpha^{A_o}_{J_c}.(W \mid U_1)$$

Again, if the reduction step $V \to V'$ is a synchronization between subprocesses in U_0, it is simple. So let us suppose that:

$$U_0 \equiv U'_0 \mid \overline{\alpha}^{B_o}_{I_c}.W_0$$

If we have $B_o \subseteq A_o \oplus J_c$ and $A_o \subseteq B_o \oplus I_c$, then

$$V' \equiv U'_0 \,|\, !\alpha^{A_o}_{J_c}.(W \mid U_1) \mid (\uparrow^{A_o \sqcup B_o}(W \mid U_1 \mid W_0))$$

We pose U' equal to:

$$U'_0 \,|\, !((\alpha^{A_o}_{J}.W) \mid (\uparrow^{A_o+J_c} U_1)) \mid (\uparrow^{A_o \sqcup B_o}(W \mid W_0))$$

We have indeed $U \to U'$ and we have $U' \sqsubseteq V'$ because $A_o \sqcup B_o \subseteq A_o + J_c$. Indeed,

$$\mathsf{Left}(A_o + J_c) \leq \max(\mathsf{Left}(A_o), \mathsf{Left}(B_o))$$

As either $J_c = J$ and so $\mathsf{Left}(A_o = J_c) = \mathsf{Left}(A_o)$, either $J_c = [I, J]$ and

$$\mathsf{Left}(A_o) + I \leq \mathsf{Right}(A_o) + I \leq \mathsf{Left}(B_o)$$

since $B_o \subseteq A_o \oplus J_c$. Moreover, we have

$$\max(\mathsf{Right}(A_o), \mathsf{Right}(B_o)) \leq \mathsf{Right}(A_o + J_c)$$

again because $B_o \subseteq A_o \oplus J_c$.

Thus, we have indeed Lemma 13, and we deduce the second point of Lemma 2.

Finally, the third point is a direct consequence of the third point. Indeed, suppose that U is reliable. So, for any reduction from U, it does not lead to an error. Let us take a reduction from V. By the third point, it gives us a reduction from U where some steps are a subtype of the steps in V. So, as an error cannot happen in the steps from U, and the only usage U such that $U \sqsubseteq \mathtt{err}$ is $\mathtt{err}$, we know that the reduction from V does not lead to an error. Thus, V is reliable. □

B. Elements of Comparison with Baillot and Ghyselen

In this section, we give intuitively a description of how to simulate types on [13] in a linear setting with usage. We say that a process has a linear use of channels if it use channel names at most one time for input and at most one time for output. For servers, we suppose that the replicated input is once and for all defined at the beginning of a process, and as free variables it can only use others servers. In their type system, a channel is given a type $\mathtt{Ch}_I(\widetilde{T})$ where I is an upper bound on the time this channel communicates. It can also be a variant of this type with only input or only output capability. Such a channel would be represented in out type system by a type $\mathtt{ch}(\widetilde{\underline{T}})/(\mathtt{In}^{[I_1,I_1]}_{J^1_c} \mid \mathtt{Out}^{I_2,I_2}_{J^2_c})$ where either J^1_c is 0 and then $I_1 \leq I$, either $J^1_c = [J_1, J_1]$ and then $I_1 + J_1 \leq I$. We have the same thing for J^2_c and I_2. To be more precise, the typing in our setting should be a non-deterministic choice (using $+$) over such usages, and the capacity should adapt to the obligation of the dual action in order to be reliable. So, for example if $I_1 \leq I_2$, then we would take: $\mathtt{ch}(\widetilde{\underline{T}})/(\mathtt{In}^{[I_1,I_1]}_{[I_2-I_1,I_2-I_1]} \mid \mathtt{Out}^{I_2,I_2}_0)$. Note that this shape of type adapts well to the way time is delayed in their setting. For example, the `tick` constructor in their setting make the time advance by 1, and in our setting, then we would obtain the usage $(\mathtt{In}^{[I_1+1,I_1+1]}_{[I_2-I_1,I_2-I_1]} \mid \mathtt{Out}^{I_2+1,I_2+1}_0)$ and we still have $I_2 - I_1 = (I_2 + 1) - (I_1 + 1)$.

In the same way, in their setting when doing an output (or input), the time is delayed by I. Here, with usages, it would be delayed by J_c which is, by definition, a delay of the shape $\uparrow^{[J,J]}$ with $J \leq I$. So, we would keep the invariant that our time annotation have the shape of singleton interval with a smaller value than the time annotation in their setting.

For servers, in the linear setting, their types have the shape: ${}_I\forall\widetilde{i}.\mathtt{serv}^J(\widetilde{T})$ where $I = 0$ is again a time annotation giving an upper bound on the time the input action of this server is defined, and J is a complexity as in our setting. So, in our setting it would be:

$$\forall\widetilde{i}.\mathtt{serv}^{[0,J]}(\widetilde{\underline{T}})/!\mathtt{In}^{[0,0]}_{\infty}.!\mathtt{Out}^{[0,\infty]}_0 \,|\, !\mathtt{Out}^{[0,\infty]}_0$$

Note that this usage is reliable. The main point here is this infinite capacity for input. Please note that because of our input rule for servers, it does not generates an infinite complexity. However, it imposes a delaying $\uparrow^{[0,\infty]}!\Gamma$ in the context. Because of the shape we gave to types, it means that the context can only have outputs for other servers as free variables, but this was the condition imposed by linearity. Note that in [13], they have a restriction on the free variables of servers that is in fact the same restriction so it does not harm the comparison to take this restriction on free variables. As an example, the bitonic sort described in [13] could be typed similarly in our setting with this kind of type.

Finally, choice in usages $U_1 + U_2$ is used to put together the different usages we obtain in the two branches of a pattern matching.

C. On the Need for Intervals in Usages

We describe informally on an example where the use of intervals is important in our work. The need for an interval capacity is apparent for the process

$$a().\overline{b}\langle\rangle \mid \mathtt{match}\ e\ \{\mathtt{case}\ \mathtt{0} \mapsto \overline{a}\langle\rangle; \mathtt{case}\ \mathtt{s}(x) \mapsto \mathtt{tick}.\overline{a}\langle\rangle\}$$

Indeed, depending on the value of e (which may be statically unknown), an output on a may be available at time 0 or 1. Thus, the input usage on a should have a capacity interval $[0, 1]$. As a result, the obligation of the output usage on b should also be an interval $[0, 1]$.

Now, one may think that we can assume that lower-bounds are always 0 (or ∞, to consider processes like Example 8) and

omit lower-bounds, since we are mainly interested in an *upper-bound* of the parallel complexity. Information about lower-bounds is, however, actually required for precise reasoning on upper-bounds. For example, consider the following process:

$$a().\overline{b}\langle\rangle \mid \texttt{tick}.\overline{a}\langle\rangle.b()$$

With intervals, a have the usage $\texttt{In}_{[1,1]}^{[0,0]} \mid \texttt{Out}_{0}^{[1,1]}$ and so b has the usage $\texttt{Out}_{[0,0]}^{[1,1]} \mid \texttt{In}_{[0,0]}^{[1,1]}$, and the parallel complexity of the process can be precisely inferred to be 1.

If we set lower-bounds to 0 and assign to a the usage $\texttt{In}_{[0,1]}^{[0,0]} \mid \texttt{Out}_{0}^{[0,1]}$ to a, then the usage of b can only be: $\texttt{Out}_{1}^{[0,1]} \mid \texttt{In}_{1}^{[0,1]}$. Note that according to the imprecise usage of a, the output on b may become ready at time 0 and then have to wait for one time unit until the input on b becomes ready; thus, the capacity of the output on b is 1, instead of $[0, 0]$. An upper-bound of the parallel complexity would therefore be inferred to be $1 + 1 = 2$ (because the usages tell us that the lefthand side process may wait for one time unit at a, and then for another time unit at b), which is too imprecise.

D. Substitution Lemma

Let us recall that the difficult cases for substitution lemma (Lemma 7) are:

1) If $\varphi; \Phi; \Gamma, b : \texttt{ch}(\widetilde{S})/W_0, c : \texttt{ch}(\widetilde{S})/W_1 \vdash P \triangleleft K$ then $\varphi; \Phi; \Gamma, b : \texttt{ch}(\widetilde{S})/(W_0 \mid W_1) \vdash P[c := b] \triangleleft K$
2) If $\varphi; \Phi; \Gamma, b{:}\forall\widetilde{i}.\texttt{serv}^K(\widetilde{S})/W_0, c{:}\forall\widetilde{i}.\texttt{serv}^K(\widetilde{S})/W_1 \vdash P \triangleleft K$ then $\varphi; \Phi; \Gamma, b : \forall\widetilde{i}.\texttt{serv}^K(\widetilde{S})/(W_0 \mid W_1) \vdash P[c := b] \triangleleft K$

For those two points, the main difficulty is for the input and output rules. We first detail the first point of this lemma, and we will detail the difference for the second point.

1) • **Case** of input, with $a \neq b$ and $a \neq c$.

$$\frac{\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{S})/W_0, c : \mathtt{ch}(\widetilde{S})/W_1, a : \mathtt{ch}(\widetilde{T})/U, \widetilde{v} : \widetilde{T} \vdash P \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} \Gamma, b : \mathtt{ch}(\widetilde{S})/(\uparrow^{J_c} W_0), c : \mathtt{ch}(\widetilde{S})/(\uparrow^{J_c} W_1), a : \mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.U \vdash a(\widetilde{v}).P \triangleleft J_c; K}$$

By induction hypothesis, we obtain $\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{S})/(W_0 \mid W_1), a : \mathtt{ch}(\widetilde{T})/U, \widetilde{v} : \widetilde{T} \vdash P[c := b] \triangleleft K$.
We then give the following proof:

$$\frac{\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{S})/(W_0 \mid W_1), a : \mathtt{ch}(\widetilde{T})/U, \widetilde{v} : \widetilde{T} \vdash P[c := b] \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} \Gamma, b : \mathtt{ch}(\widetilde{S})/(\uparrow^{J_c}(W_0 \mid W_1)), a : \mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.U \vdash a(\widetilde{v}).P \triangleleft J_c; K}$$

This case is similar to all other cases when b and c does not interfere with the typing rule. Thus, we know only show the cases when they interfere.

• **Case** of input, with $a = b$.

$$\frac{\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/U, c : \mathtt{ch}(\widetilde{T})/W_1, \widetilde{v} : \widetilde{T} \vdash P \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} \Gamma, b : \mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.U, c : \mathtt{ch}(\widetilde{T})/(\uparrow^{J_c} W_1) \vdash b(\widetilde{v}).P \triangleleft J_c; K}$$

By induction hypothesis, we obtain $\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/(U \mid W_1), \widetilde{v} : \widetilde{T} \vdash P[c := b] \triangleleft K$. So, we give the typing:

$$\frac{\dfrac{\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/(U \mid W_1), \widetilde{v} : \widetilde{T} \vdash P[c := b] \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} \Gamma, b : \mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.(U \mid W_1) \vdash b(\widetilde{v}).P[c := b] \triangleleft J_c; K}}{\varphi; \Phi; \uparrow^{J_c} \Gamma, b : \mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.U \mid (\uparrow^{J_c} W_1) \vdash b(\widetilde{v}).P[c := b] \triangleleft J_c; K}$$

Indeed, the last rule represents subtyping. This concludes this case.

• **Case** of input, with $a = c$.

$$\frac{\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/W_0, c : \mathtt{ch}(\widetilde{T})/U, \widetilde{v} : \widetilde{T} \vdash P \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} \Gamma, b : \mathtt{ch}(\widetilde{T})/(\uparrow^{J_c} W_0), c : \mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.U \vdash c(\widetilde{v}).P \triangleleft J_c; K}$$

By induction hypothesis, we obtain $\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/(W_0 \mid U), \widetilde{v} : \widetilde{T} \vdash P[c := b] \triangleleft K$. So, we give the typing:

$$\frac{\dfrac{\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/(W_0 \mid U), \widetilde{v} : \widetilde{T} \vdash P[c := b] \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} \Gamma, b : \mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.(W_0 \mid U) \vdash b(\widetilde{v}).P[c := b] \triangleleft J_c; K}}{\varphi; \Phi; \uparrow^{J_c} \Gamma, b : \mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.U \mid (\uparrow^{J_c} W_0) \vdash (c(\widetilde{v}).P)[c := b] \triangleleft J_c; K}$$

• **Case** of output, with $a = b$.

$$\frac{\varphi; \Phi; \Gamma', b : \mathtt{ch}(\widetilde{T})/V, c : \mathtt{ch}(\widetilde{T})/W_1' \vdash \widetilde{e} : \widetilde{T} \qquad \varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/U, c : \mathtt{ch}(\widetilde{T})/W_1 \vdash P \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} (\Gamma \mid \Gamma'), b : \mathtt{ch}(\widetilde{T})/\mathtt{Out}_{J_c}^{[0,0]}.(V \mid U), c : \mathtt{ch}(\widetilde{T})/\uparrow^{J_c} (W_1' \mid W_1) \vdash \overline{b}\langle\widetilde{e}\rangle.P \triangleleft J_c; K}$$

By point 1 of Lemma 7 and induction hypothesis, we obtain $\varphi; \Phi; \Gamma', b : \mathtt{ch}(\widetilde{T})/(V \mid W_1') \vdash \widetilde{e} : \widetilde{T}$ and $\varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/(U \mid W_1) \vdash P \triangleleft K$. Thus, we have:

$$\frac{\varphi; \Phi; \Gamma', b : \mathtt{ch}(\widetilde{T})/(V \mid W_1') \vdash \widetilde{e} : \widetilde{T} \qquad \varphi; \Phi; \Gamma, b : \mathtt{ch}(\widetilde{T})/(U \mid W_1) \vdash P \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} (\Gamma \mid \Gamma'), b : \mathtt{ch}(\widetilde{T})/\mathtt{Out}_{J_c}^{[0,0]}.(V \mid U \mid W_1 \mid W_1') \vdash \overline{b}\langle\widetilde{e}\rangle.P \triangleleft J_c; K}$$

$$\varphi; \Phi; \uparrow^{J_c} (\Gamma \mid \Gamma'), b : \mathtt{ch}(\widetilde{T})/\mathtt{Out}_{J_c}^{[0,0]}.(V \mid U) \mid \uparrow^{J_c} (W_1 \mid W_1') \vdash \overline{b}\langle\widetilde{e}\rangle.P \triangleleft J_c; K$$

Again, the last rule is obtained by subtyping. We have a similar proof for the case $a = c$.

2) We know work on the case of servers. The notations are a bit cumbersome but the proofs are similar to the one for channels. The only point that need some details is for server input as there is the replication in usages that appear.

• **Case** of input, with $a \neq b$ and $a \neq c$.

$$\frac{(\varphi, \widetilde{i}); \Phi; \Gamma, a : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/U, b : \forall \widetilde{j}.\mathtt{serv}^{K'}(\widetilde{S})/W_0, c : \forall \widetilde{j}.\mathtt{serv}^{K'}(\widetilde{S})/W_1, \widetilde{v} : \widetilde{T} \vdash P \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} !\Gamma, a : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/!\mathtt{In}_{J_c}^{[0,0]}.U, b : \forall \widetilde{j}.\mathtt{serv}^K(')/\widetilde{S}(\uparrow^{J_c} !W_0), c : \forall \widetilde{j}.\mathtt{serv}^{K'}(\widetilde{S})/(\uparrow^{J_c} !W_1) \vdash !a(\widetilde{v}).P \triangleleft [0,0]}$$

By induction hypothesis, we have $(\varphi, \widetilde{i}); \Phi; \Gamma, a{:}\forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/U, b{:}\forall \widetilde{j}.\mathtt{serv}^{K'}(\widetilde{S})/(W_0 \mid W_1), \widetilde{v}{:}\widetilde{T} \vdash P[c := b] \triangleleft K$.
So, we have the proof

$$\frac{(\varphi, \widetilde{i}); \Phi; \Gamma, a : \forall \widetilde{i}.\mathtt{serv}^{K'}(\widetilde{T})/U, b : \forall \widetilde{j}.\mathtt{serv}^{K'}(\widetilde{S})/(W_0 \mid W_1), \widetilde{v} : \widetilde{T} \vdash P[c := b] \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} !\Gamma, a : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/!\mathtt{In}_{J_c}^{[0,0]}.U, b : \forall \widetilde{j}.\mathtt{serv}^{K'}(\widetilde{S})/(\uparrow^{J_c} !(W_0 \mid W_1)) \vdash !a(\widetilde{v}).P[c := b] \triangleleft [0,0]}$$

• **Case** of input, with $a = b$.

$$\frac{(\varphi, \widetilde{i}); \Phi; \Gamma, b : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/U, c : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/W_1, \widetilde{v} : \widetilde{T} \vdash P \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} !\Gamma, b : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/!\mathtt{In}_{J_c}^{[0,0]}.U, c : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/(\uparrow^{J_c} !W_1) \vdash !a(\widetilde{v}).P \triangleleft [0,0]}$$

By induction hypothesis, we obtain $(\varphi, \widetilde{i}); \Phi; \Gamma, b : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/(U \mid W_1), \widetilde{v} : \widetilde{T} \vdash P \triangleleft K$

$$\frac{\dfrac{(\varphi, \widetilde{i}); \Phi; \Gamma, b : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/(U \mid W_1), \widetilde{v} : \widetilde{T} \vdash P[c := b] \triangleleft K}{\varphi; \Phi; \uparrow^{J_c} !\Gamma, b : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/!\mathtt{In}_{J_c}^{[0,0]}.(U \mid W_1) \vdash !a(\widetilde{v}).P[c := b] \triangleleft [0,0]}}{\varphi; \Phi; \uparrow^{J_c} !\Gamma, b : \forall \widetilde{i}.\mathtt{serv}^K(\widetilde{T})/(!\mathtt{In}_{J_c}^{[0,0]}.U \mid \uparrow^{J_c} !W_1) \vdash !a(\widetilde{v}).P[c := b] \triangleleft [0,0]}$$

This last derivation is obtained by subtyping. Indeed, by definition we have $\uparrow^{J_c}!W_1 = !\uparrow^{J_c}W_1$. Then,

$$!\mathtt{In}_{J_c}^{[0,0]}.U \,|\, !\uparrow^{J_c}W_1 \equiv !(\mathtt{In}_{J_c}^{[0,0]}.U \mid \uparrow^{J_c}W_1) \sqsubseteq !\mathtt{In}_{J_c}^{[0,0]}.(U \mid W_1)$$

- The case $a = c$ is similar to the previous one.

This concludes the proof.

E. Congruence Equivalence

Proof. Let us show Lemma 9. We prove this by induction on $P \equiv Q$. Note that for a process P, the typing system is not syntax-directed because of the subtyping rule. However, by reflexivity and transitivity of subtyping, we can always assume that a proof has exactly *one* subtyping rule before any syntax-directed rule. Moreover, notice that in those kinds of proof, the top-level rule of subtyping can be ignored. Indeed, we can always simulate exactly the same subtyping rule for both P and Q We first show this propriety for base case of congruence. The reflexivity is trivial then we have those interesting cases:

- **Case** $(\nu a)P \mid Q \equiv (\nu a)(P \mid Q)$ with a not free in Q. Suppose $\varphi; \Phi; \Gamma \mid \Delta \vdash (\nu a)P \mid Q \triangleleft K$. Then the proof has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\varphi;\Phi;\Gamma', a : T \vdash P \triangleleft K_1'} \quad T \text{ reliable}}{\varphi;\Phi;\Gamma' \vdash (\nu a)P \triangleleft K_1'} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \Gamma'; K_1' \subseteq K_1}{\varphi;\Phi;\Gamma \vdash (\nu a)P \triangleleft K_1} \quad \dfrac{\pi'}{\varphi;\Phi;\Delta \vdash Q \triangleleft K_2}}{\varphi;\Phi;\Gamma \mid \Delta \vdash (\nu a)P \mid Q \triangleleft K_1 \sqcup K_2}$$

  By weakening (Lemma 4), we obtain a proof π_w' of $\varphi; \Phi; \Delta, a : (T/0) \vdash Q \triangleleft K_2$. Thus, we have the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\varphi;\Phi;\Gamma', a : T \vdash P \triangleleft K_1'} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \Gamma'; K_1' \subseteq K_1}{\varphi;\Phi;\Gamma, a : T \vdash P \triangleleft K_1} \quad \dfrac{\pi_w'}{\varphi;\Phi;\Delta, a : (T/0) \vdash Q \triangleleft K_2}}{\varphi;\Phi;\Gamma \mid \Delta, a : T \vdash P \mid Q \triangleleft K_1 \sqcup K_2} \quad T \text{ reliable}}{\varphi;\Phi;\Gamma \mid \Delta \vdash (\nu a)(P \mid Q) \triangleleft K_1 \sqcup K_2}$$

  For the converse, suppose $\varphi; \Phi; \Gamma \vdash (\nu a)(P \mid Q) \triangleleft K$. Then the proof has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\varphi;\Phi;\Gamma_P, a : T_P \vdash P \triangleleft K_1} \quad \dfrac{\pi'}{\varphi;\Phi;\Gamma_Q, a : T_Q \vdash Q \triangleleft K_2}}{\varphi;\Phi;\Gamma_P \mid \Gamma_Q, a : T_P \mid T_Q \vdash P \mid Q \triangleleft K_1 \sqcup K_2} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \Gamma_P \mid \Gamma_Q; T \sqsubseteq T_P \mid T_Q; K_1 \sqcup K_2 \subseteq K}{\varphi;\Phi;\Gamma, a : T \vdash P \mid Q \triangleleft K} \quad T \text{ reliable}}{\varphi;\Phi;\Gamma \vdash (\nu a)(P \mid Q) \triangleleft K}$$

  Since a is not free in Q, by Lemma 5, from π' we obtain a proof π_s' of $\varphi; \Phi; \Gamma_Q \vdash Q \triangleleft K_2$. We then derive the following typing:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\varphi;\Phi;\Gamma_P, a : T_P \vdash P \triangleleft K_1} \quad \varphi;\Phi \vdash T \sqsubseteq T_P \mid T_Q \sqsubseteq T_P}{\varphi;\Phi;\Gamma_P, a : T \vdash P \triangleleft K_1} \quad T \text{ reliable}}{\varphi;\Phi;\Gamma_P \vdash (\nu a)P \triangleleft K_1} \quad \dfrac{\pi_s'}{\varphi;\Phi;\Gamma_Q \vdash Q \triangleleft K_2}}{\varphi;\Phi;\Gamma_P \mid \Gamma_Q \vdash (\nu a)P \mid Q \triangleleft K_1 \sqcup K_2} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \Gamma_P \mid \Gamma_Q; K_1 \sqcup K_2 \subseteq K}{\varphi;\Phi;\Gamma \vdash (\nu a)P \mid Q \triangleleft K}$$

- **Case** $m : (P \mid Q) \equiv m : P \mid m : Q$. Suppose $\varphi; \Phi; \uparrow^{[m,m]}\Gamma \vdash m : (P \mid Q) \triangleleft K + [m, m]$. Then we have:

$$\dfrac{\dfrac{\dfrac{\pi_P}{\varphi;\Phi;\Gamma_P \vdash P \triangleleft K_1} \quad \dfrac{\pi_Q}{\varphi;\Phi;\Gamma_Q \vdash Q \triangleleft K_2}}{\varphi;\Phi;\Gamma_P \mid \Gamma_Q \vdash (P \mid Q) \triangleleft K_1 \sqcup K_2} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \Gamma_P \mid \Gamma_Q; K_1 \sqcup K_2 \subseteq K}{\varphi;\Phi;\Gamma \vdash (P \mid Q) \triangleleft K}}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma \vdash m : (P \mid Q) \triangleleft K + [m, m]}$$

  By Lemma 2, from $\varphi; \Phi \vdash \Gamma \sqsubseteq \Gamma_P \mid \Gamma_Q$ we obtain $\varphi; \Phi \vdash \uparrow^{[m,m]}\Gamma \sqsubseteq (\uparrow^{[m,m]}\Gamma_P) \mid (\uparrow^{[m,m]}\Gamma_Q)$. So, we give the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{\varphi;\Phi;\Gamma_P \vdash P \triangleleft K_1}}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma_P \vdash m : P \triangleleft K_1 + [m, m]} \quad \dfrac{\dfrac{\pi_Q}{\varphi;\Phi;\Gamma_Q \vdash Q \triangleleft K_2}}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma_Q \vdash m : Q \triangleleft K_2 + [m, m]}}{\varphi;\Phi;(\uparrow^{[m,m]}\Gamma_P) \mid (\uparrow^{[m,m]}\Gamma_Q) \vdash m : P \mid m : Q \triangleleft (K_1 \sqcup K_2) + [m, m]} \quad \varphi;\Phi \vdash \uparrow^{[m,m]}\Gamma \sqsubseteq (\uparrow^{[m,m]}\Gamma_P) \mid (\uparrow^{[m,m]}\Gamma_Q)}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma \vdash m : P \mid m : Q \triangleleft (K_1 \sqcup K_2) + [m, m]} \quad \varphi;\Phi \vDash K_1 \sqcup K_2 \subseteq K}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma \vdash m : P \mid m : Q \triangleleft K + [m, m]}$$

  Now, suppose we have a typing $\varphi; \Phi; \Gamma_P \mid \Gamma_Q \vdash m : P \mid m : Q \triangleleft K_1 \sqcup K_2$. The typing has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{\varphi;\Phi;\Delta_P \vdash P \triangleleft K_P}}{\varphi;\Phi;\uparrow^{[m,m]}\Delta_P \vdash m : P \triangleleft K_P + [m,m]}}{\varphi;\Phi;\Gamma_P \vdash m : P \triangleleft K_1} \qquad \dfrac{\dfrac{\dfrac{\pi_Q}{\varphi;\Phi;\Delta_Q \vdash Q \triangleleft K_Q}}{\varphi;\Phi;\uparrow^{[m,m]}\Delta_Q \vdash m : Q \triangleleft K_Q + [m,m]}}{\varphi;\Phi;\Gamma_Q \vdash m : Q \triangleleft K_2}}{\varphi;\Phi;\Gamma_P \mid \Gamma_Q \vdash m : P \mid m : Q \triangleleft K_1 \sqcup K_2}$$

with

$$\varphi;\Phi \vdash \Gamma_P \sqsubseteq \uparrow^{[m,m]}\Delta_P \qquad \varphi;\Phi \vdash \Gamma_Q \sqsubseteq \uparrow^{[m,m]}\Delta_Q \qquad \varphi;\Phi \vDash K_P + [m,m] \subseteq K_1 \qquad \varphi;\Phi \vDash K_Q + [m,m] \subseteq K_2$$

So, we derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{\varphi;\Phi;\Delta_P \vdash P \triangleleft K_P} \qquad \dfrac{\pi_Q}{\varphi;\Phi;\Delta_Q \vdash Q \triangleleft K_Q}}{\varphi;\Phi;\Delta_P \mid \Delta_Q \vdash (P \mid Q) \triangleleft K_P \sqcup K_Q}}{\varphi;\Phi;\uparrow^{[m,m]}(\Delta_P \mid \Delta_Q) \vdash m : (P \mid Q) \triangleleft (K_P \sqcup K_Q) + [m,m] \qquad \varphi;\Phi \vdash \Gamma_P \mid \Gamma_Q \sqsubseteq \uparrow^{[m,m]}(\Delta_P \mid \Delta_Q); (K_P \sqcup K_Q) + [m,m] \subseteq K_1 \sqcup K_2}}{\varphi;\Phi;\Gamma_P \mid \Gamma_Q \vdash m : (P \mid Q) \triangleleft K}$$

This concludes this case.

- **Case** $m : (\nu a)P \equiv (\nu a)(m : P)$.
  Suppose $\varphi;\Phi;\uparrow^{[m,m]}\Gamma \vdash m : (\nu a)P \triangleleft K + [m,m]$. Then, the typing has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\varphi;\Phi;\Gamma', a : T \vdash P \triangleleft K'} \quad T \text{ reliable}}{\varphi;\Phi;\Gamma' \vdash (\nu a)P \triangleleft K'} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \Gamma'; K' \subseteq K}{\varphi;\Phi;\Gamma \vdash (\nu a)P \triangleleft K}}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma \vdash m : (\nu a)P \triangleleft K + [m,m]}$$

  By Lemma 3, we know that $\uparrow^{[m,m]}T$ is reliable. So, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\varphi;\Phi;\Gamma', a : T \vdash P \triangleleft K'} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \Gamma'; K' \subseteq K}{\varphi;\Phi;\Gamma, a : T \vdash P \triangleleft K}}{\varphi;\Phi;\uparrow^{[m,m]}(\Gamma, a : T) \vdash (m : P) \triangleleft K + [m,m]} \quad \uparrow^{[m,m]}T \text{ reliable}}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma \vdash (\nu a)(m : P) \triangleleft K + [m,m]}$$

  For the converse, suppose we have $\varphi;\Phi;\Gamma \vdash (\nu a)(m : P) \triangleleft K$. Then, the typing has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\varphi;\Phi;\Gamma', a : T' \vdash P \triangleleft K'}}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma', a : \uparrow^{[m,m]}T' \vdash (m : P) \triangleleft K' + [m,m]} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \uparrow^{[m,m]}\Gamma'; T \sqsubseteq \uparrow^{[m,m]}T'; K' + [m,m] \subseteq K}{\varphi;\Phi;\Gamma, a : T \vdash (m : P) \triangleleft K} \quad T \text{ reliable}}{\varphi;\Phi;\Gamma \vdash (\nu a)(m : P) \triangleleft K}$$

  As T is reliable, by Lemma 2, we have $\uparrow^{[m,m]}T'$ reliable. Then, by Lemma 3, we have T' reliable. So, we give the typing:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\varphi;\Phi;\Gamma', a : T' \vdash P \triangleleft K'} \quad T' \text{ reliable}}{\varphi;\Phi;\Gamma' \vdash (\nu a)P \triangleleft K'}}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma' \vdash m : (\nu a)P \triangleleft K' + [m,m]} \quad \varphi;\Phi \vdash \Gamma \sqsubseteq \uparrow^{[m,m]}\Gamma'; K' + [m,m] \subseteq K}{\varphi;\Phi;\Gamma \vdash m : (\nu a)P \triangleleft K}$$

This concludes the interesting base case. Symmetry and transitivity are direct, and for the cases of contextual congruence, the proof is straightforward. □

F. Subject Reduction

We now give elements for Theorem 1.

Again, when considering the typing of P, the first subtyping rule has no importance. We now proceed by doing the case analysis on the rules of Figure 2. In order to simplify the proof, we will also consider that types and indexes invariant by subtyping (like the complexity in a server) are not renamed with subtyping. Note that this only add cumbersome notations but it does not change the core of the proof.

- **Case** $(n :!a(\widetilde{v}).P) \mid (m : \overline{a}\langle\widetilde{e}\rangle.Q) \Rightarrow (n :!a(\widetilde{v}).P) \mid (\max(m,n) : (P[\widetilde{v} := \widetilde{e}] \mid Q))$. Consider the typing $\varphi;\Phi;\Gamma_0 \mid \Delta_0, a : \forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/(U_0 \mid V_0) \vdash (n :!a(\widetilde{v}).P) \mid (m : \overline{a}\langle\widetilde{e}\rangle.Q) \triangleleft K_0 \sqcup K_0'$. The first rule is the rule for parallel composition, then the proof is split into the two following subtree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{(\varphi,\widetilde{i});\Phi;\Gamma_2,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/U_2,\widetilde{v}:\widetilde{T}\vdash P\triangleleft K_a}}{\varphi;\Phi;\uparrow^{J_c}!\Gamma_2,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/!\mathtt{In}^{[0,0]}_{J_c}.U_2\vdash !a(\widetilde{v}).P\triangleleft[0,0]}\quad \varphi;\Phi\vdash\Gamma_1\sqsubseteq\uparrow^{J_c}!\Gamma_2;U_1\sqsubseteq!\mathtt{In}^{[0,0]}_{J_c}.U_2;[0,0]\subseteq K_1}{\varphi;\Phi;\Gamma_1,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/U_1\vdash !a(\widetilde{v}).P\triangleleft K_1}}{\varphi;\Phi;\uparrow^{[n,n]}\Gamma_1,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/\uparrow^{[n,n]}U_1\vdash n:!a(\widetilde{v}).P\triangleleft K_1+[n,n]}\quad \varphi;\Phi\vdash\Gamma_0\sqsubseteq\uparrow^{[n,n]}\Gamma_1;U_0\sqsubseteq\uparrow^{[n,n]}U_1;K_1+[n,n]\subseteq K_0}{\varphi;\Phi;\Gamma_0,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/U_0\vdash n:!a(\widetilde{v}).P\triangleleft K_0}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_e}{\varphi;\Phi;\Delta_2,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/V_2\vdash\widetilde{e}:\widetilde{T}\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}}\quad\dfrac{\pi_Q}{\varphi;\Phi;\Delta_2',a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/V_2'\vdash Q\triangleleft K_2}}{\varphi;\Phi;\uparrow^{J_c'}(\Delta_2\mid\Delta_2'),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/\mathtt{Out}^{[0,0]}_{J_c'}.(V_2\mid V_2')\vdash\overline{a}\langle\widetilde{e}\rangle.Q\triangleleft J_c';(K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\})}\quad(1)}{\varphi;\Phi;\Delta_1,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/V_1\vdash\overline{a}\langle\widetilde{e}\rangle.Q\triangleleft K_1'}}{\varphi;\Phi;\uparrow^{[m,m]}\Delta_1,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/\uparrow^{[m,m]}V_1\vdash m:\overline{a}\langle\widetilde{e}\rangle.Q\triangleleft K_1'+[m,m]}\quad\varphi;\Phi\vdash\Delta_0\sqsubseteq\uparrow^{[n,n]}\Delta_1;V_0\sqsubseteq\uparrow^{[m,m]}V_1;K_1'+[m,m]\subseteq K_0'}{\varphi;\Phi;\Delta_0,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/V_0\vdash m:\overline{a}\langle\widetilde{e}\rangle.Q\triangleleft K_0'}$$

where

$$(1)\qquad\varphi;\Phi\vdash\Delta_1\sqsubseteq\uparrow^{J_c'}(\Delta_2\mid\Delta_2')\qquad\varphi;\Phi\vdash V_1\sqsubseteq\mathtt{Out}^{[0,0]}_{J_c'}.(V_2\mid V_2')\qquad\varphi;\Phi\vDash J_c';(K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\})\subseteq K_1'$$

First, by the index substitution lemma (Lemma 6), from π_P we obtain a proof:

$$\pi_P\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}:\qquad\varphi;\Phi;\Gamma_2,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/U_2,\widetilde{v}:\widetilde{T}\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}\vdash P\triangleleft K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}$$

Since the index variables $\widetilde{i}$ can only be free in $\widetilde{T}$ and K_a.
Then, we know that $\Gamma_0\mid\Delta_0$ is defined. Moreover, we have

$$\varphi;\Phi\vdash\Gamma_0\sqsubseteq\uparrow^{[n,n]}\Gamma_1\qquad\varphi;\Phi\vdash\Gamma_1\sqsubseteq\uparrow^{J_c}!\Gamma_2\qquad\varphi;\Phi\vdash\Delta_0\sqsubseteq\uparrow^{[m,m]}\Delta_1\qquad\Delta_1\sqsubseteq\uparrow^{J_c'}(\Delta_2\mid\Delta_2')$$

So, for the channel and server types, in those seven contexts, the shape of the type does not change (only the usage can change). Let us look at base types. For a context Γ, we write Γ^{Nat} the restriction of Γ to base types. Then, we have:

$$\Gamma_0^{\mathsf{Nat}}=\Delta_0^{\mathsf{Nat}}\qquad\varphi;\Phi\vdash\Gamma_0^{\mathsf{Nat}}\sqsubseteq\Gamma_1^{\mathsf{Nat}}\sqsubseteq\Gamma_2^{\mathsf{Nat}}\qquad\varphi;\Phi\vdash\Delta_0^{\mathsf{Nat}}\sqsubseteq\Delta_1^{\mathsf{Nat}}\sqsubseteq\Delta_2^{\mathsf{Nat}}\qquad\Delta_2^{\mathsf{Nat}}=\Delta_2'^{\mathsf{Nat}}$$

Similarly, we note Γ^ν the restriction of a context to its channel and server types. Thus, we have $\Gamma=\Gamma^\nu,\Gamma^{\mathsf{Nat}}$.
So, from π_e and $\pi_P\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}$ we obtain by subtyping:

$$\pi_P':\qquad\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\Gamma_2^\nu,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/U_2,\widetilde{v}:\widetilde{T}\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}\vdash P\triangleleft K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}$$

$$\pi_e':\qquad\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\Delta_2^\nu,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/V_2\vdash\widetilde{e}:\widetilde{T}\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}$$

So, we use the substitution lemma (Lemma 7) and we obtain:

$$\pi_{sub}:\qquad\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},(\Gamma_2^\nu\mid\Delta_2^\nu),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/(U_2\mid V_2)\vdash P[\widetilde{v}:=\widetilde{e}]\triangleleft K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}$$

As previously, by subtyping from π_Q, we have:

$$\pi_Q':\qquad\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\Delta_2'^\nu,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/V_2'\vdash Q\triangleleft K_2$$

Thus, with the parallel composition rule (as parallel composition of context is defined) and subtyping we have:

$$\pi_{PQ}:\qquad\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},(\Gamma_2^\nu\mid\Delta_2^\nu\mid\Delta_2'^\nu),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/(U_2\mid V_2\mid V_2')\vdash(P[\widetilde{v}:=\widetilde{e}]\mid Q)\triangleleft K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}$$

Let us denote $M=\max(m,n)$. Thus, we derive the proof:

$$\dfrac{\dfrac{\pi_{PQ}}{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},(\Gamma_2^\nu\mid\Delta_2^\nu\mid\Delta_2'^\nu),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/(U_2\mid V_2\mid V_2')\vdash(P[\widetilde{v}:=\widetilde{e}]\mid Q)\triangleleft K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}}}{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\uparrow^{[M,M]}(\Gamma_2^\nu\mid\Delta_2^\nu\mid\Delta_2'^\nu),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/\uparrow^{[M,M]}(U_2\mid V_2\mid V_2')\vdash M:(P[\widetilde{v}:=\widetilde{e}]\mid Q)\triangleleft(K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\})+[M,M]}$$

Now, recall that by hypothesis, $U_0\mid V_0$ is reliable. We have:

$$\varphi;\Phi\vdash U_0\sqsubseteq\uparrow^{[n,n]}U_1\qquad\varphi;\Phi\vdash U_1\sqsubseteq!\mathtt{In}^{[0,0]}_{J_c}.U_2\qquad\varphi;\Phi\vdash V_0\sqsubseteq\uparrow^{[m,m]}V_1\qquad\varphi;\Phi\vdash V_1\sqsubseteq\mathtt{Out}^{[0,0]}_{J_c'}(V_2\mid V_2')$$

So, by Point 1 of Lemma 2, with transitivity and parallel composition of subusage, we have:

$$\varphi;\Phi\vdash U_0\mid V_0\sqsubseteq(\uparrow^{[n,n]}U_1)\mid(\uparrow^{[m,m]}V_1)\sqsubseteq!\mathtt{In}_{J_c}^{[n,n]}.U_2\mid\mathtt{Out}_{J_c'}^{[m,m]}(V_2\mid V_2')$$

By Point 3 of Lemma 2, we have $!\mathtt{In}_{J_c}^{[n,n]}.U_2\mid\mathtt{Out}_{J_c'}^{[m,m]}(V_2\mid V_2')$ reliable. So, in particular, we have:

$$\varphi;\Phi\vdash!\mathtt{In}_{J_c}^{[n,n]}.U_2\mid\mathtt{Out}_{J_c'}^{[m,m]}(V_2\mid V_2')\longrightarrow!\mathtt{In}_{J_c}^{[n,n]}.U_2\mid\uparrow^{[M,M]}(U_2\mid V_2\mid V_2')$$

$$\varphi;\Phi\vDash[n,n]\subseteq[m,m]\oplus J_c'\qquad\varphi;\Phi\vDash[m,m]\subseteq[n,n]\oplus J_c$$

Thus, we deduce immediately that neither J_c or J_c' are $[\infty,\infty]$ and that

$$\varphi;\Phi\vDash[M,M]\subseteq[m,M]\subseteq[n,n]+J_c\qquad\varphi;\Phi\vDash[M,M]\subseteq[n,M]\subseteq[m,m]+J_c'$$

So, we have in particular, with Lemma 10 and Point 1 of Lemma 2 and parallel composition:

$$\varphi;\Phi\vdash\Gamma_0\mid\Delta_0\sqsubseteq(\uparrow^{[n,n]}\Gamma_1)\mid\uparrow^{[m,m]}\Delta_1\sqsubseteq(\uparrow^{[n,n]+J_c}!\Gamma_2)\mid(\uparrow^{[m,m]+J_c'}(\Delta_2\mid\Delta_2'))\sqsubseteq\uparrow^{[M,M]}(!\Gamma_2\mid\Delta_2\mid\Delta_2')$$

We also have

$$\varphi;\Phi\vDash(K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\})+[M,M]\subseteq(J_c';(K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\})+[m,m])\subseteq K_0'$$

As Thus, we simplify a bit the derivation given above, and we have:

$$\dfrac{\dfrac{\dfrac{\pi_{PQ}}{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},(\Gamma_2^\nu\mid\Delta_2^\nu\mid\Delta_2'^\nu),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/(U_2\mid V_2\mid V_2')\vdash(P[\widetilde{v}:=\widetilde{e}]\mid Q)\triangleleft K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}}}{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\uparrow^{[M,M]}(\Gamma_2^\nu\mid\Delta_2^\nu\mid\Delta_2'^\nu),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/\uparrow^{[M,M]}(U_2\mid V_2\mid V_2')\vdash M:(P[\widetilde{v}:=\widetilde{e}]\mid Q)\triangleleft(K_2\sqcup K_a\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\})+[M,M]}}{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\uparrow^{[M,M]}(\Gamma_2^\nu\mid\Delta_2^\nu\mid\Delta_2'^\nu),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/\uparrow^{[M,M]}(U_2\mid V_2\mid V_2')\vdash M:(P[\widetilde{v}:=\widetilde{e}]\mid Q)\triangleleft K_0'}$$

We also have the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{(\varphi,\widetilde{i});\Phi;\Gamma_2,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/U_2,\widetilde{v}:\widetilde{T}\vdash P\triangleleft K_a}}{\varphi;\Phi;\uparrow^{J_c}!\Gamma_2,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/!\mathtt{In}_{J_c}^{[0,0]}.U_2\vdash!a(\widetilde{v}).P\triangleleft[0,0]}}{\varphi;\Phi;\uparrow^{[n,n]+J_c}!\Gamma_2,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/!\mathtt{In}_{J_c}^{[n,n]}.U_2\vdash n:!a(\widetilde{v}).P\triangleleft[n,n]}}{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\uparrow^{[M,M]}!\Gamma_2^\nu,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/!\mathtt{In}_{J_c}^{[n,n]}.U_2\vdash n:!a(\widetilde{v}).P\triangleleft K_0}$$

So, by parallel composition of those two derivation we obtain a proof of:

$$\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\uparrow^{[M,M]}(!\Gamma_2^\nu\mid\Gamma_2^\nu\mid\Delta_2^\nu\mid\Delta_2'^\nu),a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/!\mathtt{In}_{J_c}^{[n,n]}.U_2\mid(\uparrow^{[M,M]}(U_2\mid V_2\mid V_2'))\vdash(n:!a(\widetilde{v}).P)\mid M:(P[\widetilde{v}:=\widetilde{e}]\mid Q)\triangleleft K_0\sqcup K_0'$$

By Point 2 of Lemma 2, there exists W such that:

$$\varphi;\Phi\vdash U_0\mid V_0\longrightarrow^*W\qquad\varphi;\Phi\vdash W\sqsubseteq!\mathtt{In}_{J_c}^{[n,n]}.U_2\mid\uparrow^{[M,M]}(U_2\mid V_2\mid V_2')$$

So, by subtyping we have a proof:

$$\varphi;\Phi;\Gamma_0^{\mathsf{Nat}},\Gamma_0^\nu\mid\Delta_0^\nu,a:\forall\widetilde{i}.\mathtt{serv}^{K_a}(\widetilde{T})/W\vdash(n:!a(\widetilde{v}).P)\mid M:(P[\widetilde{v}:=\widetilde{e}]\mid Q)\triangleleft K_0\sqcup K_0'$$

This concludes this case.

- **Case** $(n:a(\widetilde{v}).P)\mid(m:\overline{a}\langle\widetilde{e}\rangle.Q)\Rightarrow(\max(m,n):(P[\widetilde{v}:=\widetilde{e}]\mid Q))$. Consider the typing $\varphi;\Phi;\Gamma_0\mid\Delta_0,a:\mathtt{ch}(\widetilde{T})/(U_0\mid V_0)\vdash(n:a(\widetilde{v}).P)\mid(m:\overline{a}\langle\widetilde{e}\rangle.Q)\triangleleft K_0\sqcup K_0'$. The first rule is the rule for parallel composition, then the proof is split into the two following subtree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{\varphi;\Phi;\Gamma_2,a:\mathtt{ch}(\widetilde{T})/U_2,\widetilde{v}:\widetilde{T}\vdash P\triangleleft K_2}}{\varphi;\Phi;\uparrow^{J_c}\Gamma_2,a:\mathtt{ch}(\widetilde{T})/\mathtt{In}_{J_c}^{[0,0]}.U_2\vdash a(\widetilde{v}).P\triangleleft J_c;K_2\qquad\varphi;\Phi\vdash\Gamma_1\sqsubseteq\uparrow^{J_c}\Gamma_2;U_1\sqsubseteq\mathtt{In}_{J_c}^{[0,0]}.U_2;J_c;K_2\subseteq K_1}}{\varphi;\Phi;\Gamma_1,a:\mathtt{ch}(\widetilde{T})/U_1\vdash a(\widetilde{v}).P\triangleleft K_1}}{\varphi;\Phi;\uparrow^{[n,n]}\Gamma_1,a:\mathtt{ch}(\widetilde{T})/\uparrow^{[n,n]}U_1\vdash n:a(\widetilde{v}).P\triangleleft K_1+[n,n]\qquad\varphi;\Phi\vdash\Gamma_0\sqsubseteq\uparrow^{[n,n]}\Gamma_1;U_0\sqsubseteq\uparrow^{[n,n]}U_1;K_1+[n,n]\subseteq K_0}}{\varphi;\Phi;\Gamma_0,a:\mathtt{ch}(\widetilde{T})/U_0\vdash n:a(\widetilde{v}).P\triangleleft K_0}$$

$$\dfrac{\dfrac{\dfrac{\pi_e}{\varphi;\Phi;\Delta_2, a:\mathtt{ch}(\widetilde{T})/V_2 \vdash \widetilde{e}:\widetilde{T}} \quad \dfrac{\pi_Q}{\varphi;\Phi;\Delta_2', a:\mathtt{ch}(\widetilde{T})/V_2' \vdash Q \triangleleft K_2'}}{\varphi;\Phi;\uparrow^{J_c'}(\Delta_2 \mid \Delta_2'), a:\mathtt{ch}(\widetilde{T})/\mathtt{Out}_{J_c'}^{[0,0]}.(V_2 \mid V_2') \vdash \overline{a}\langle\widetilde{e}\rangle.Q \triangleleft J_c'; K_2' \qquad (1)}}{\varphi;\Phi;\Delta_1, a:\mathtt{ch}(\widetilde{T})/V_1 \vdash \overline{a}\langle\widetilde{e}\rangle.Q \triangleleft K_1'}}{\dfrac{\varphi;\Phi;\uparrow^{[m,m]}\Delta_1, a:\mathtt{ch}(\widetilde{T})/\uparrow^{[m,m]}V_1 \vdash m:\overline{a}\langle\widetilde{e}\rangle.Q \triangleleft K_1'+[m,m] \qquad \varphi;\Phi \vdash \Delta_0 \sqsubseteq \uparrow^{[n,n]}\Delta_1; V_0 \sqsubseteq \uparrow^{[m,m]}V_1; K_1'+[m,m] \subseteq K_0'}{\varphi;\Phi;\Delta_0, a:\mathtt{ch}(\widetilde{T})/V_0 \vdash m:\overline{a}\langle\widetilde{e}\rangle.Q \triangleleft K_0'}}$$

where

$$(1) \qquad \varphi;\Phi \vdash \Delta_1 \sqsubseteq \uparrow^{J_c'}(\Delta_2 \mid \Delta_2') \qquad \varphi;\Phi \vdash V_1 \sqsubseteq \mathtt{Out}_{J_c'}^{[0,0]}.(V_2 \mid V_2') \qquad \varphi;\Phi \vDash J_c'; K_2' \subseteq K_1'$$

First, we know that $\Gamma_0 \mid \Delta_0$ is defined. Moreover, we have

$$\varphi;\Phi \vdash \Gamma_0 \sqsubseteq \uparrow^{[n,n]}\Gamma_1 \qquad \varphi;\Phi \vdash \Gamma_1 \sqsubseteq \uparrow^{J_c}\Gamma_2 \qquad \varphi;\Phi \vdash \Delta_0 \sqsubseteq \uparrow^{[m,m]}\Delta_1 \qquad \Delta_1 \sqsubseteq \uparrow^{J_c'}(\Delta_2 \mid \Delta_2')$$

So, for the channel and server types, in those seven contexts, the shape of the type does not change (only the usage can change). We also have:

$$\Gamma_0^{\mathsf{Nat}} = \Delta_0^{\mathsf{Nat}} \qquad \varphi;\Phi \vdash \Gamma_0^{\mathsf{Nat}} \sqsubseteq \Gamma_1^{\mathsf{Nat}} \sqsubseteq \Gamma_2^{\mathsf{Nat}} \qquad \varphi;\Phi \vdash \Delta_0^{\mathsf{Nat}} \sqsubseteq \Delta_1^{\mathsf{Nat}} \sqsubseteq \Delta_2^{\mathsf{Nat}} \qquad \Delta_2^{\mathsf{Nat}} = \Delta_2'^{\mathsf{Nat}}$$

So, from π_e and π_P we obtain by subtyping:

$$\varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, \Gamma_2^{\nu}, a:\mathtt{ch}(\widetilde{T})/U_2, \widetilde{v}:\widetilde{T} \vdash P \triangleleft K_2 \qquad \varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, \Delta_2^{\nu}, a:\mathtt{ch}(\widetilde{T})/V_2 \vdash \widetilde{e}:\widetilde{T}$$

So, we use the substitution lemma (Lemma 7) and we obtain:

$$\varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, (\Gamma_2^{\nu} \mid \Delta_2^{\nu}), a:\mathtt{ch}(\widetilde{T})/(U_2 \mid V_2) \vdash P[\widetilde{v} := \widetilde{e}] \triangleleft K_2$$

As previously, by subtyping from π_Q, we have:

$$\varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, \Delta_2'^{\nu}, a:\mathtt{ch}(\widetilde{T})/V_2' \vdash Q \triangleleft K_2'$$

Thus, with the parallel composition rule (as parallel composition of context is defined) and subtyping we have:

$$\varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, (\Gamma_2^{\nu} \mid \Delta_2^{\nu} \mid \Delta_2'^{\nu}), a:\mathtt{ch}(\widetilde{T})/(U_2 \mid V_2 \mid V_2') \vdash (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_2 \sqcup K_2'$$

Let us denote $M = \max(m, n)$. Thus, we derive the proof:

$$\dfrac{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, (\Gamma_2^{\nu} \mid \Delta_2^{\nu} \mid \Delta_2'^{\nu}), a:\mathtt{ch}(\widetilde{T})/(U_2 \mid V_2 \mid V_2') \vdash (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_2 \sqcup K_2'}{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, \uparrow^{[M,M]}(\Gamma_2^{\nu} \mid \Delta_2^{\nu} \mid \Delta_2'^{\nu}), a:\mathtt{ch}(\widetilde{T})/\uparrow^{[M,M]}(U_2 \mid V_2 \mid V_2') \vdash M:(P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft (K_2 \sqcup K_2') + [M,M]}$$

Now, recall that by hypothesis, $U_0 \mid V_0$ is reliable. We have:

$$\varphi;\Phi \vdash U_0 \sqsubseteq \uparrow^{[n,n]}U_1 \qquad \varphi;\Phi \vdash U_1 \sqsubseteq \mathtt{In}_{J_c}^{[0,0]}.U_2 \qquad \varphi;\Phi \vdash V_0 \sqsubseteq \uparrow^{[m,m]}V_1 \qquad \varphi;\Phi \vdash V_1 \sqsubseteq \mathtt{Out}_{J_c'}^{[0,0]}(V_2 \mid V_2')$$

So, by Point 1 of Lemma 2, with transitivity and parallel composition of subusage, we have:

$$\varphi;\Phi \vdash U_0 \mid V_0 \sqsubseteq (\uparrow^{[n,n]}U_1) \mid (\uparrow^{[m,m]}V_1) \sqsubseteq \mathtt{In}_{J_c}^{[n,n]}.U_2 \mid \mathtt{Out}_{J_c'}^{[m,m]}(V_2 \mid V_2')$$

By Point 3 of Lemma 2, we have $\mathtt{In}_{J_c}^{[n,n]}.U_2 \mid \mathtt{Out}_{J_c'}^{[m,m]}(V_2 \mid V_2')$ reliable. So, in particular, we have:

$$\varphi;\Phi \vdash \mathtt{In}_{J_c}^{[n,n]}.U_2 \mid \mathtt{Out}_{J_c'}^{[m,m]}(V_2 \mid V_2') \longrightarrow \uparrow^{[M,M]}(U_2 \mid V_2 \mid V_2')$$

$$\varphi;\Phi \vDash [n,n] \subseteq [m,m] \oplus J_c' \qquad \varphi;\Phi \vDash [m,m] \subseteq [n,n] \oplus J_c$$

Thus, we deduce that

$$\varphi;\Phi \vDash [M,M] \subseteq [n,n] + J_c \qquad \varphi;\Phi \vDash [M,M] \subseteq [m,m] + J_c'$$

So, we have in particular, with Lemma 10 and Point 1 of Lemma 2 and parallel composition:

$$\varphi;\Phi \vdash \Gamma_0 \mid \Delta_0 \sqsubseteq (\uparrow^{[n,n]}\Gamma_1) \mid \uparrow^{[m,m]}\Delta_1 \sqsubseteq (\uparrow^{[n,n]+J_c}\Gamma_2) \mid (\uparrow^{[m,m]+J_c'}(\Delta_2 \mid \Delta_2')) \sqsubseteq \uparrow^{[M,M]}(\Gamma_2 \mid \Delta_2 \mid \Delta_2')$$

We also have

$$\varphi;\Phi \vDash K_2 + [M,M] \subseteq J_c; K_2 + [n,n] \subseteq K_0 \qquad \varphi;\Phi \vDash K_2' + [M,M] \subseteq J_c'; K_2' + [m,m] \subseteq K_0'$$

So, we obtain direclty $\varphi;\Phi \vDash (K_2 \sqcup K_2') + [M,M] \subseteq K_0 \sqcup K_0'$

Thus, we can simplify a bit the derivation given above, and we have:

$$\dfrac{\dfrac{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, (\Gamma_2^{\nu} \mid \Delta_2^{\nu} \mid \Delta_2'^{\nu}), a : \mathtt{ch}(\widetilde{T})/(U_2 \mid V_2 \mid V_2') \vdash (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_2 \sqcup K_2'}{\varphi;\Phi;\Gamma_0^{\mathsf{Nat}}, \uparrow^{[M,M]}(\Gamma_2^{\nu} \mid \Delta_2^{\nu} \mid \Delta_2'^{\nu}), a : \mathtt{ch}(\widetilde{T})/\uparrow^{[M,M]}(U_2 \mid V_2 \mid V_2') \vdash M : (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft (K_2 \sqcup K_2') + [M,M]}}{\varphi;\Phi;(\Gamma_0 \mid \Delta_0), a : \mathtt{ch}(\widetilde{T})/\uparrow^{[M,M]}(U_2 \mid V_2 \mid V_2') \vdash M : (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_0 \sqcup K_0'}$$

By Point 2 of Lemma 2, there exists W such that:

$$\varphi;\Phi \vdash U_0 \mid V_0 \longrightarrow^* W \qquad \varphi;\Phi \vdash W \sqsubseteq \uparrow^{[M,M]}(U_2 \mid V_2 \mid V_2')$$

So, by subtyping we have a proof:

$$\varphi;\Phi;\Gamma_0 \mid \Delta_0, a : \mathtt{ch}(\widetilde{T})/W \vdash M : (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_0 \sqcup K_0'$$

This concludes this case.

- **Case** $\mathtt{match}\ \mathtt{s}(e)\ \{\mathtt{case}\ \mathtt{0} \mapsto P; \mathtt{case}\ \mathtt{s}(x) \mapsto Q\} \Rightarrow Q[x := e]$. The case for an expression equals to 0 is similar, so we only present this one. Suppose given a derivation $\mathtt{match}\ \mathtt{s}(e)\ \{\mathtt{case}\ \mathtt{0} \mapsto P; \mathtt{case}\ \mathtt{s}(x) \mapsto Q\} \triangleleft K$. Then the proof has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_e}{\varphi;\Phi;\Delta \vdash e : \mathsf{Nat}[I',J']}}{\varphi;\Phi;\Delta \vdash \mathtt{s}(e) : \mathsf{Nat}[I'+1,J'+1]} \qquad \varphi;\Phi \vdash \Gamma \sqsubseteq \Delta; \mathsf{Nat}[I'+1,J'+1] \sqsubseteq \mathsf{Nat}[I,J]}{\varphi;\Phi;\Gamma \vdash \mathtt{s}(e) : \mathsf{Nat}[I,J]} \qquad \pi_P \qquad \pi_Q}{\mathtt{match}\ \mathtt{s}(e)\ \{\mathtt{case}\ \mathtt{0} \mapsto P; \mathtt{case}\ \mathtt{s}(x) \mapsto Q\} \triangleleft K}$$

Where π_Q is a proof of $\phi;(\Phi, J \geq 1);\Gamma, x : \mathsf{Nat}[I\text{–}1][J\text{–}1] \vdash Q \triangleleft K$, and π_P is a typing derivation for P that does not interest us in this case.

By definition of subtyping, we have:

$$\varphi;\Phi \vDash I \leq I' + 1 \qquad \varphi;\Phi \vDash J' + 1 \leq J$$

From this, we deduce the following constraints:

$$\varphi;\Phi \vDash J \geq 1 \qquad \varphi;\Phi \vDash I\text{–}1 \leq I' \qquad \varphi;\Phi \vDash J' \leq J\text{–}1$$

Thus, with the subtyping rule and the proof π_e we obtain:

$$\varphi;\Phi;\Delta \vdash e : \mathsf{Nat}[I\text{–}1, J\text{–}1]$$

Then, by Lemma 5, from π_Q we obtain a proof of $\varphi;\Phi;\Gamma, x : \mathsf{Nat}[I\text{–}1][J\text{–}1] \vdash Q \triangleleft K$. By the substitution lemma (Lemma 7), we obtain $\varphi;\Phi;\Gamma \vdash Q[x := e] \triangleleft K$. This concludes this case.

- **Case** $n : P \Rightarrow n : Q$ with $P \Rightarrow Q$. Suppose that $\varphi;\Phi;\uparrow^{[n,n]}\Gamma \vdash n : P \triangleleft K + [n,n]$. Then, the proof has the shape:

$$\dfrac{\varphi;\Phi;\Gamma \vdash P \triangleleft K}{\varphi;\Phi;\uparrow^{[n,n]}\Gamma \vdash n : P \triangleleft K + [n,n]}$$

By Lemma 3, if $\uparrow^{[n,n]}\Gamma$ is reliable then Γ is reliable. By induction hypothesis, we have a proof $\varphi;\Phi;\Gamma' \vdash Q \triangleleft K$ with $\varphi;\Phi \vdash \Gamma \longrightarrow^* \Gamma'$.

We give the proof:

$$\dfrac{\varphi;\Phi;\Gamma' \vdash Q \triangleleft K}{\varphi;\Phi;\uparrow^{[n,n]}\Gamma' \vdash n : Q \triangleleft K + [n,n]}$$

And we have indeed $\varphi;\Phi \vdash \uparrow^{[n,n]}\Gamma \longrightarrow^* \uparrow^{[n,n]}\Gamma'$ by Lemma 3.

- **Case** $P \Rightarrow Q$ with $P \equiv P'$, $P' \Rightarrow Q'$ and $Q \equiv Q'$. Suppose that $\varphi;\Phi;\Gamma \vdash P \triangleleft K$. By Lemma 9, we have $\varphi;\Phi;\Gamma \vdash P' \triangleleft K$. By induction hypothesis, we obtain $\varphi;\Phi;\Gamma' \vdash Q' \triangleleft K$ with $\varphi;\Phi \vdash \Gamma \longrightarrow^* \Gamma'$. Then, again by Lemma 9, we have $\varphi;\Phi;\Gamma' \vdash Q \triangleleft K$. This concludes this case.

This concludes the proof of Theorem 1.