Types for Complexity of Parallel Computation in Pi-calculus (Long Version)

Patrick Baillot, Alexis Ghyselen

Types for Complexity of Parallel Computation in Pi-calculus (Technical Report)

Patrick Baillot, Alexis Ghyselen

Univ Lyon, CNRS, ENS de Lyon, Universite Claude-Bernard Lyon 1, LIP, F-69342, Lyon Cedex 07, France

Abstract

Type systems as a technique to analyse or control programs have been extensively studied for functional programming languages. In particular some systems allow to extract from a typing derivation a complexity bound on the program. We explore how to extend such results to parallel complexity in the setting of the pi-calculus, considered as a communication-based model for parallel computation. We study two notions of time complexity: the total computation time without parallelism (the work) and the computation time under maximal parallelism (the span). We define operational semantics to capture those two notions, and present two type systems from which one can extract a complexity bound on a process. The type systems are inspired both by size types and by input/output types, with additional temporal information about communications.

Contents

1 Introduction **1**

2 Pi-calculus with input/output types **3**

3 Size Types and Parallel Complexity **6**
3.1 Parallel Complexity : the Span 6
3.2 Size Types with Temporal Information 8
3.3 Intermediate Lemmas 11
3.4 Subject Reduction 13

4 An example : Bitonic Sort **22**

5 Work of a Process **24**

6 A Hint for Type Inference **26**
6.1 The Restricted Type System 27

7 Parallel Complexity and Causal Complexity **31**
7.1 Causal Complexity 31
7.2 Parallel Complexity $\geq$ Causal Complexity 33
7.3 Causal Complexity $\geq$ Parallel Complexity 34

1 Introduction

The problem of certifying time complexity bounds for programs is a challenging question, related to the problem of statically infering time complexity, and it has been extensively studied in the setting of sequential programming languages. One particular approach to these questions is that of type systems, which offers the advantage of providing an analysis which is formally-grounded, compositional

and modular. In the functional framework several rich type systems have been proposed, such that if a program can be assigned a type, then one can extract from the type derivation a complexity bound for its execution on any input (see e.g. [13, 17, 14, 12, 3, 2]). The type system itself thus provides a complexity certification procedure, and if a type inference algorithm is also provided one obtains a complexity inference procedure. This research area is also related to implicit computational complexity, which aims at providing type systems or static criteria to characterize some complexity classes within a programming language (see e.g. [16, 8, 19, 10, 9]), and which have sometimes in a second step inspired a complexity certification or inference procedure.

However, while the topic of complexity certification has been thoroughly investigated for sequential programs both for space and time bounds, there only have been a few contributions in the settings of parallel programs and distributed systems. In these contexts, several notions of cost can be of interest to abstract the computation time. First one can wish to know what is during a program execution the total cumulated computation time on all processors. This is called the *work* of the program. Second, one can wonder if an infinite number of processors were available, what would be the execution time of the program when it is maximally parallelized. This is called the *span* or *depth* of the program.

The paper [15] has addressed the problem of analysing the time complexity of programs written in a parallel first-order functional language. In this language one can spawn computations in parallel and use the resulting values in the body of the program. This allows to express a large bunch of classical parallel algorithms. Their approach is based on amortized complexity and builds on a line of work in the setting of sequential languages to define type systems, which allow to derive bounds on the work and the span of the program. However the language they are investigating does not include more elaborate synchronization features.

Our goal is to provide an approach to analyse the time complexity of programs written in a rich language for communication-based parallel computation, allowing the representation of several synchronization features. We use for that π-calculus, a process calculus which provides process creation, channel name creation and name-passing in communication. An alternative approach could be to use session types, as in [4, 5]. However we choose here to explore this question for π-calculus because it is in general more expressive than session types. We want to propose methods that, given a parallel program written in π-calculus, allow to derive upper bounds on its work and span. Let us mention that the notions of work and span are not only of theoretical interest. Some classical results provide upper bounds, expressed by means of the work (w) and span (s), on the evaluation time of a parallel program on a given number p of processors. For instance such a program can be evaluated on a shared-multiprocessor system (SMP) with p processors in time $O(\max(w/p, s))$ (see e.g. [11]).

Our goal in this paper is essentially fundamental and methodological, in the sense that we aim at proposing type systems which are general enough, well-behaved and provide good complexity properties. We do not focus yet at this stage on the design and efficiency of type inference algorithms.

We want to be able to derive complexity bounds which are parametric in the size of inputs, for instance which depend on the length of a list. For that it will be useful to have a language of types that can carry information about sizes, and for this reason we take inspiration from size types. So data-types will be annotated with an index which will provide some information on the size of values. Our approach then follows the standard approach to typing in the π-calculus, namely typing a channel by providing the types of the messages that can be sent or received through it. Actually a second ingredient will be necessary for us, input/output types. In this setting a channel is given a set of capabilities: it can be an input, an output, or have both input/output capabilities.

Contributions. The contributions of this paper are then the following ones. We consider a π-calculus with an explicit `tick` construction; this allows to specify several cost models, instead of only counting the number of reduction steps. We propose two semantics of this π-calculus to define formally and in a simple way the work and the span of a process. We then design two type systems for the π-calculus, one for the work and one for the span, and establish for both a soundness theorem: if a process is well-typed in the first (resp. second) type system, then its type provides an expression which, for its execution on any input, bounds the work (resp. span). We illustrate our typing with the example of the parallel bitonic sort.

Discussion. Note that even though one of the main usages of π-calculus is to specify and analyse concurrent systems, the present paper does not aim at analysing the complexity of π-calculus concurrent programs. Indeed, some typical examples of concurrent systems like semaphores will simply not be typable in the system for span, because of linearity conditions. As explained above, our interest here

is instead focused on parallel computation expressed in the π-calculus, which can include some form of cooperative concurrency. We believe the analysis of complexity bounds for concurrent π-calculus is another challenging question, which we want to address in future work.

Acknowledgement. We are grateful to Naoki Kobayashi for suggesting the definition of annotated processes and their reduction that we use in this paper (see Section 3.1)

2 Pi-calculus with input/output types

We present here a classical π-calculus with input/output types. More detail about those types or π-calculus can be found in [20]. The set of *variables*, *expressions* and *pre-processes* are defined by the following grammar.

$$\begin{aligned} v &:= x, y, z \mid a, b, c \\ e &:= v \mid \mathtt{0} \mid \mathtt{s}(e) \mid [] \mid e :: e' \mid \mathtt{tt} \mid \mathtt{ff} \\ P, Q &:= 0 \mid P \mid Q \mid !a(\tilde{v}).P \mid a(\tilde{v}).P \mid \overline{a}\langle \tilde{e} \rangle \mid (\nu a)P \mid \mathtt{match}(e)\ \{\mathtt{0} \mapsto P;;\ \mathtt{s}(x) \mapsto Q\} \\ &\quad \mid \mathtt{match}(e)\ \{[] \mapsto P;;\ x :: y \mapsto Q\} \end{aligned}$$

Variables x, y, z denote *base type variables*, so they represent integers or lists. Variables a, b, c denote *channel variables*. The notation $\tilde{v}$ stands for a sequence of variables $v_1, v_2, \ldots, v_k$. In the same way, $\tilde{e}$ is a sequence of expressions. We work up to α-renaming, and we use $P[\tilde{v} := \tilde{e}]$ to denote the substitution of the free variables $\tilde{v}$ in P by $\tilde{e}$.

We define on those pre-processes a congruence relation $\equiv$, such that this relation is the least congruence relation closed under:

$$P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$$

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

Note that the last rule can always be made by renaming. Also, note that contrary to other congruence relation for the π-calculus, we do not consider the rule for replicated input ($!P \equiv !P|P$), and α-conversion is not an explicit rule in the congruence. With this definition, we can give a canonical form for pre-processes, as in [18].

Definition 1 (Guarded Pre-processes and Canonical Form)**.** *A pre-process G is* guarded *if it is an input, a replicated input, an output or a pattern matching. We say that a pre-process is in* canonical form *if it has the form*

$$(\nu \tilde{a})(G_1 \mid \cdots \mid G_n)$$

with $G_1, \ldots, G_n$ that are guarded pre-processes.

And we now show that all processes have a somewhat unique canonical form.

Lemma 1 (Existence of Canonical Form)**.** *For any pre-process P, there is a Q in canonical form such that $P \equiv Q$.*

Proof. Le us suppose that, by renaming, all the introduction of new variables have different names and that they also differ from the free variables already in P. We can then proceed by induction on the structure of P. The only interesting case is for parallel composition. Suppose that

$$P \equiv (\nu \tilde{a})(P_1 \mid \cdots \mid P_n) \qquad Q \equiv (\nu \tilde{b})(Q_1 \mid \cdots \mid Q_m)$$

With $P_1, \ldots, P_n, Q_1, \ldots, Q_m$ guarded pre-processes. Then, by hypothesis on the name of variables, we have $\tilde{a}$ and $\tilde{b}$ disjoint and $\tilde{a}$ is not free in Q, as well as $\tilde{b}$ is not free in P. So, we obtain

$$P \mid Q \equiv (\nu \tilde{a})(\nu \tilde{b})(P_1 \mid \cdots \mid P_n \mid Q_1 \mid \cdots \mid Q_m)$$

□

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

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

$$\frac{}{\texttt{match}([])\ \{[] \mapsto P;;\ x :: y \mapsto Q\} \to P} \qquad \frac{}{\texttt{match}(e :: e')\ \{[] \mapsto P;;\ x :: y \mapsto Q\} \to Q[x, y := e, e']}$$

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

Figure 1: Reduction Rules

$$\frac{}{\mathcal{B} \sqsubseteq \mathcal{B}} \qquad \frac{\tilde{T} \sqsubseteq \tilde{U} \qquad \tilde{U} \sqsubseteq \tilde{T}}{\mathsf{ch}(\tilde{T}) \sqsubseteq \mathsf{ch}(\tilde{U})} \qquad \frac{}{\mathsf{ch}(\tilde{T}) \sqsubseteq \mathsf{in}(\tilde{T})} \qquad \frac{}{\mathsf{ch}(\tilde{T}) \sqsubseteq \mathsf{out}(\tilde{T})}$$

$$\frac{\tilde{T} \sqsubseteq \tilde{U}}{\mathsf{in}(\tilde{T}) \sqsubseteq \mathsf{in}(\tilde{U})} \qquad \frac{\tilde{U} \sqsubseteq \tilde{T}}{\mathsf{out}(\tilde{T}) \sqsubseteq \mathsf{out}(\tilde{U})} \qquad \frac{T \sqsubseteq T' \qquad T' \sqsubseteq T''}{T \sqsubseteq T''}$$

Figure 2: Subtyping Rules

Lemma 2 (Uniqueness of Canonical Form)**.** *If*

$$(\nu\tilde{a})(P_1 \mid \cdots \mid P_n) \equiv (\nu\tilde{b})(Q_1 \mid \cdots \mid Q_m)$$

with $P_1, \ldots, P_n, Q_1, \ldots, Q_m$ guarded pre-processes, then $m = n$ and $\tilde{a}$ is a permutation of $\tilde{b}$. Moreover, for some permutation $Q'_1, \ldots, Q'_n$ of $Q_1, \ldots, Q_n$, we have $P_i \equiv Q'_i$ for all i.

Proof. Recall that α-renaming is not a rule of $\equiv$, otherwise this propriety would be false. As before, we suppose that all names are already well-chosen. Then, let us define a set $\texttt{name}$ of channel variable and a multiset $\texttt{gp}$ of guarded pre-processes.

- $\texttt{name}(0) = \emptyset$ and $\texttt{gp}(0) = \emptyset$.
- $\texttt{name}(P \mid Q) = \texttt{name}(P) \coprod \texttt{name}(Q)$ and $\texttt{gp}(P \mid Q) = \texttt{gp}(P) + \texttt{gp}(Q)$.
- $\texttt{name}(P) = \emptyset$ and $\texttt{gp}(P) = [P]$, when P is guarded.
- $\texttt{name}((\nu a)P) = \texttt{name}(\texttt{P}) \coprod \{a\}$ and $\texttt{gp}((\nu a)P) = \texttt{gp}(P)$.

Then, we can easily show the following lemma by definition of the congruence relation.

Lemma 3. *If $P \equiv Q$ then $\texttt{name}(P) = \texttt{name}(Q)$ and if $\texttt{gp}(P) = [P_1, \ldots, P_n]$ and $\texttt{gp}(Q) = [Q_1, \ldots, Q_m]$, then $m = n$ and for some permutation $Q'_1, \ldots, Q'_n$ of $Q_1, \ldots, Q_n$, we have $P_i \equiv Q'_i$ for all i.*

Finally, Lemma 2 is a direct consequence of Lemma 3. □

We can now define the reduction relation $P \to Q$ for pre-processes. It is defined by the rules given in Figure 1. Remark that substitution should be well-defined in order to do the reduction. For example, $(a(\tilde{v}).P)[a := \texttt{0}]$ is not a valid substitution, as a channel variable is replaced by an integer. More formally, channel variables must be substituted by other channel variables and base type variables can be substituted by any expression except channel variables. However, when we have typed pre-processes, this yields well-typed substitutions.

The set of *base types* and types are given by the following grammar.

$$\mathcal{B} := \mathsf{Nat} \mid \mathsf{List}(\mathcal{B}) \qquad T := \mathcal{B} \mid \mathsf{ch}(\tilde{T}) \mid \mathsf{in}(\tilde{T}) \mid \mathsf{out}(\tilde{T})$$

When a type T is not a base type, we call it a *channel type*. Then, we define a subtyping relation on those types, expressed by the rules of Figure 2

Definition 2 (Typing Contexts)**.** *A* typing context Γ *is a sequence of hypotheses of the form $x : \mathcal{B}$ or $a : T$ where T is a channel type.*

$$\frac{v : T \in \Gamma}{\Gamma \vdash v : T} \qquad \frac{}{\Gamma \vdash \mathsf{0} : \mathsf{Nat}} \qquad \frac{\Gamma \vdash e : \mathsf{Nat}}{\Gamma \vdash \mathtt{s}(e) : \mathsf{Nat}} \qquad \frac{}{\Gamma \vdash [] : \mathsf{List}(\mathcal{B})}$$

$$\frac{\Gamma \vdash e : \mathcal{B} \qquad \Gamma \vdash e' : \mathsf{List}(\mathcal{B})}{\Gamma \vdash e :: e' : \mathsf{List}(\mathcal{B})} \qquad \frac{\Delta \vdash e : U \qquad \Gamma \sqsubseteq \Delta \qquad U \sqsubseteq T}{\Gamma \vdash e : T}$$

Figure 3: Typing Rules for Expressions

$$\frac{}{\Gamma \vdash 0} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \frac{\Gamma \vdash a : \mathsf{in}(\tilde{T}) \qquad \Gamma, \tilde{v} : \tilde{T} \vdash P}{\Gamma \vdash !a(\tilde{v}).P}$$

$$\frac{\Gamma \vdash a : \mathsf{in}(\tilde{T}) \qquad \Gamma, \tilde{v} : \tilde{T} \vdash P}{\Gamma \vdash a(\tilde{v}).P} \qquad \frac{\Gamma \vdash a : \mathsf{out}(\tilde{T}) \qquad \Gamma \vdash \tilde{e} : \tilde{T}}{\Gamma \vdash \overline{a}\langle \tilde{e} \rangle} \qquad \frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\nu a)P}$$

$$\frac{\Gamma \vdash e : \mathsf{Nat} \qquad \Gamma \vdash P \qquad \Gamma, x : \mathsf{Nat} \vdash Q}{\Gamma \vdash \mathtt{match}(e)\ \{\mathsf{0} \mapsto P;;\ \mathtt{s}(x) \mapsto Q\}} \qquad \frac{\Gamma \vdash e : \mathsf{List}(\mathcal{B}) \qquad \Gamma \vdash P \qquad \Gamma, x : \mathcal{B}, y : \mathsf{List}(\mathcal{B}) \vdash Q}{\Gamma \vdash \mathtt{match}(e)\ \{[] \mapsto P;;\ x :: y \mapsto Q\}}$$

$$\frac{\Delta \vdash P \qquad \Gamma \sqsubseteq \Delta}{\Gamma \vdash P}$$

Figure 4: Typing Rules for Pre-Processes

We can now define typing for expressions and pre-processes. This is expressed by the rules of Figure 3 and Figure 4.

Finally, we can now show the following lemma.

Lemma 4 (Closed Typed Normal Forms)**.** *A pre-process P such that $\vdash P$ is in normal form for $\rightarrow$ if and only if*

$$P \equiv (\nu(a_1, \ldots a_n))(!b_1(\tilde{v_1^0}).P_1 \mid \cdots \mid !b_k(\tilde{v_k^0}).P_k \mid c_1(\tilde{v_1^1}).Q_1 \mid \cdots \mid c_{k'}(\tilde{v_{k'}^1}).Q_{k'} \mid \overline{d_1}\langle \tilde{e_1} \rangle \mid \cdots \mid \overline{d_{k''}}\langle \tilde{e_{k''}} \rangle)$$

with $(\{b_i \mid 1 \leq i \leq k\} \cup \{c_i \mid 1 \leq i \leq k'\}) \cap \{d_i \mid 1 \leq i \leq k\} = \emptyset$.

Proof. In order to show that, we first give an exhaustive list of possibilities for a reduction, as in [18].

Lemma 5. *If $R \rightarrow R'$ then one of the following statements is true (where $R_1, \ldots, R_n$ are guarded pre-processes)*

- $$R \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid !a(\tilde{v}).P \mid \overline{a}\langle \tilde{e} \rangle) \qquad R' \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid !a(\tilde{v}).P \mid P[\tilde{v} := \tilde{e}])$$
- $$R \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid a(\tilde{v}).P \mid \overline{a}\langle \tilde{e} \rangle) \qquad R' \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid P[\tilde{v} := \tilde{e}])$$
- $$R \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid \mathtt{match}(\mathsf{0})\ \{\mathsf{0} \mapsto P;;\ \mathtt{s}(x) \mapsto Q\}) \qquad R' \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid P)$$
- $$R \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid \mathtt{match}(\mathtt{s}(e))\ \{\mathsf{0} \mapsto P;;\ \mathtt{s}(x) \mapsto Q\}) \qquad R' \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid Q[x := e])$$
- $$R \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid \mathtt{match}([])\ \{[] \mapsto P;;\ x :: y \mapsto Q\}) \qquad R' \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid P)$$
- $$R \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid \mathtt{match}(e :: e')\ \{[] \mapsto P;;\ x :: y \mapsto Q\}) \qquad R' \equiv (\nu \tilde{b})(R_1 \mid \cdots \mid R_n \mid Q[x, y := e, e'])$$

Proof. By induction on $R \to R'$. All base cases are straightforward. Then, in parallel composition, we use Lemma 1 to obtain the correct form. The contextual rule for ν is straightforward, and finally, the reduction up to congruence is straightforward using the transitivity of $\equiv$. □

We can now show Lemma 4. Suppose that

$$P \equiv (\nu(a_1, \dots a_n))(!b_1(\tilde{v_1^0}).P_1 \mid \cdots \mid !b_k(\tilde{v_k^0}).P_k \mid c_1(\tilde{v_1^1}).Q_1 \mid \cdots \mid c_{k'}(\tilde{v_{k'}^1}).Q_{k'} \mid \overline{d_1}\langle \tilde{e_1} \rangle \mid \cdots \mid \overline{d_{k''}}\langle \tilde{e_{k''}} \rangle)$$

with $(\{b_i \mid 1 \leq i \leq k\} \cup \{c_i \mid 1 \leq i \leq k'\}) \cap \{d_i \mid 1 \leq i \leq k\} = \emptyset$. By Lemma 2, this canonical form for P is unique. As P cannot have any of the possible form described in Lemma 5, P cannot be reduced by $\to$ thus P is indeed in normal form for $\to$.

Conversely, suppose that P is in normal form for $\to$, with $\vdash P$. Let us write the canonical form:

$$P \equiv (\nu \tilde{a})(P_1 \mid \cdots \mid P_n)$$

First, let us show that there is pattern matching in $P_1, \dots, P_n$. Indeed, if P_i has the form $\texttt{match}(e)\ \{\texttt{0} \mapsto R; ;\ \texttt{s}(x) \mapsto R'\}$, then by typing, we have $\tilde{a} : \tilde{T} \vdash \texttt{match}(e)\ \{\texttt{0} \mapsto R; ;\ \texttt{s}(x) \mapsto R'\}$. Thus, we obtain $\tilde{a} : \tilde{T} \vdash e : \mathsf{Nat}$. By definition of expressions, as all type in $\tilde{T}$ must be channel types, we have e an actual integer, thus P would not be in normal form for $\to$. Then, the canonical form can be written:

$$(\nu(a_1, \dots a_n))(!b_1(\tilde{v_1^0}).P_1 \mid \cdots \mid !b_k(\tilde{v_k^0}).P_k \mid c_1(\tilde{v_1^1}).Q_1 \mid \cdots \mid c_{k'}(\tilde{v_{k'}^1}).Q_{k'} \mid \overline{d_1}\langle \tilde{e_1} \rangle \mid \cdots \mid \overline{d_{k''}}\langle \tilde{e_{k''}} \rangle)$$

Now suppose, that $a \in ((\{b_i \mid 1 \leq i \leq k\} \cup \{c_i \mid 1 \leq i \leq k'\}) \cap \{d_i \mid 1 \leq i \leq k\})$. In the type derivation $\vdash P$, a is given a channel type T. As a consequence, in the (replicated) input and in the output, the type of $\tilde{v}$ in the input and $\tilde{e}$ in the output corresponds, thus the substitution is well-defined and so P is reducible. This is absurd, thus $(\{b_i \mid 1 \leq i \leq k\} \cup \{c_i \mid 1 \leq i \leq k'\}) \cap \{d_i \mid 1 \leq i \leq k\}) = \emptyset$, and we obtain the desired form.

This concludes the proof of Lemma 4. □

In the following, we will use a generalized version of Lemma 4, with exactly the same proof.

3 Size Types and Parallel Complexity

We enrich the previous set of pre-processes with a new constructor: $\texttt{tick}.P$. This new set of terms is called the set of *processes*. Intuitively, $\texttt{tick}.P$ stands for "after one unit of time, the process continues as P". We extend the congruence relation and typing to this new constructor, thus we add the following rule for congruence and for typing.

$$\frac{P \equiv Q}{\texttt{tick}.P \equiv \texttt{tick}.Q} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash \texttt{tick}.P}$$

A process of the form $\texttt{tick}.P$ is considered a guarded process. Moreover, $\texttt{tick}.P$ should be considered as a stuck process for the reduction $\to$. For example, the process $a(\tilde{v}).P \mid \texttt{tick}.\overline{a}\langle \tilde{e} \rangle$ is not reducible for the relation $\to$. In order to reduce the tick, we define another reduction relation that stands for "one unit of time", thus, this new relation will be linked with our notion of complexity for our calculus.

3.1 Parallel Complexity : the Span

An interesting notion of complexity in this calculus is the parallel one. A possible way to define such a parallel complexity would be to take causal complexity [8, 7, 6], however we believe there is a simpler presentation for our case. The idea has been proposed by Naoki Kobayashi (private communication). 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 have the meaning P with m ticks before.

We define a new congruence relation $\equiv$ on those processes, defined by:

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

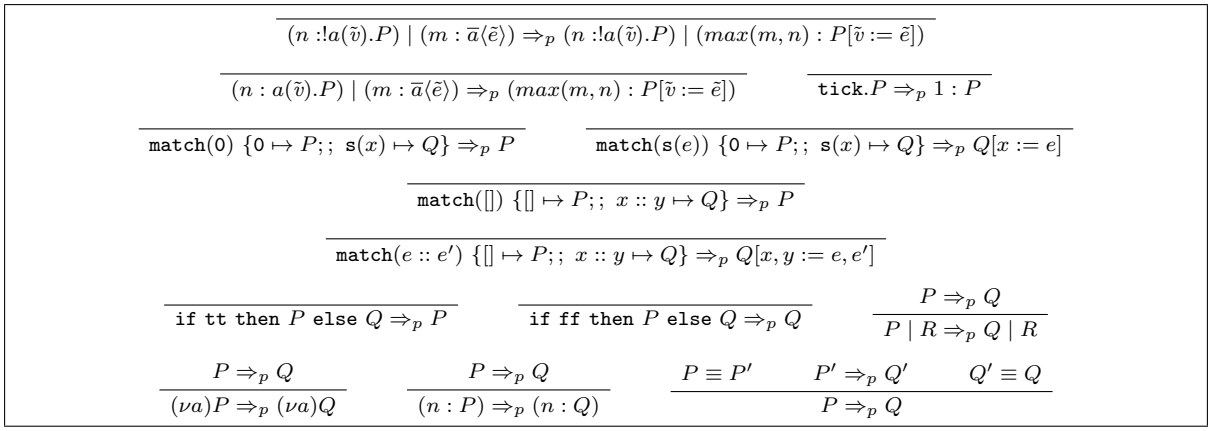

Figure 5: Reduction Rules

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

$$m : (P \mid Q) \equiv m : P \mid m : Q \qquad m : (\nu a)P \equiv (\nu a)(m : P) \qquad m : (n : P) \equiv (m + n) : P \qquad 0 : P \equiv P$$

The last line means intuitively that the ticks can be distributed over parallel composition (this is because we consider parallel complexity), then name creation can be put before ticks when necessary, and finally, succession of ticks can be grouped together.

With this congruence relation and this new constructor, we can give a new shape to the canonical form presented in Definition 1.

Definition 3 (Canonical Form for Annotated Processes)**.** *An annotated process is in canonical form if it has the shape:*

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

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

Remark that the congruence relation above allows to obtain this canonical form from any annotated processes. With this intuition in mind, we can then define a reduction relation $\Rightarrow_p$ for annotated processes. The rules are given in Figure 5. We do not detail the rule for integers as they are deducible from the one for lists. Intuitively, this semantics works as the usual semantics for pi-calculus, but when doing a synchronization, we keep the maximal annotation, and ticks are memorized in the annotations.

We can then define the parallel complexity of an annotated process.

Definition 4 (Parallel Complexity)**.** *Let P be an annotated process. We define its* local complexity *$\mathcal{C}_\ell(P)$ by:*

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

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

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 $max(n \mid P(\Rightarrow_p)^* Q \wedge \mathcal{C}_\ell(Q) = n)$ where $(\Rightarrow_p)^*$ is the reflexive and transitive closure of $\Rightarrow_p$.*

To show is that this parallel complexity is well-behaved, we need the two following lemmas.

Lemma 6 (Congruence and Local Complexity)**.** *Let P, P' be annotated processes such that $P \equiv P'$. Then, we have $\mathcal{C}_\ell(P) = \mathcal{C}_\ell(P')$.*

The proof can be done by induction on the congruence relation. All the base cases are direct, and all context rules works directly by induction hypothesis.

Lemma 7. *Let P, P' be annotated processes such that $P \Rightarrow_p P'$. Then, we have $\mathcal{C}_\ell(P') \geq \mathcal{C}_\ell(P)$*

The proof can be done by induction on $\Rightarrow_p$. The main point is that guarded processes have a local complexity equal to 0, so doing a reduction will always increment this local complexity.

With this two lemmas, we can see that in order to obtain the complexity of an annotated process, we need to reduce it with $\Rightarrow_p$ and then for each possible normal form, we have to read its local complexity, and we take the maximum over all normal forms. Moreover, this semantics respects the conditions given in the beginning of this section.

3.2 Size Types with Temporal Information

We will now define a type system to bound the span of a process. The goal is to obtain a soundness result: if a process P is typable then we can derive an integer K such that the global complexity of P is bounded by K.

Our type system relies on the definition of indices that give more information about the type. Those indices were for example used in [3] in the non-concurrent case. We also enrich type with temporal information, following the idea of [4] to obtain complexity bound.

Definition 5. *The set of* indices *for natural number is given by the following grammar.*

$$I, J, K := i, j, k \mid f(I_1, \ldots, I_n)$$

The variables i, j, k are called index variables. The set of index variables is denoted $\mathcal{V}$. We suppose given a set of function symbol containing for example the addition and the multiplication. We assume that each function symbol f comes with an interpretation $[\![f]\!] : \mathbb{N}^{\mathtt{ar}(f)} \to \mathbb{N}$.

Given an index valuation $\rho : \mathcal{V} \to \mathbb{N}$, we extend the interpretation of function symbols to indices, noted $[\![I]\!]_\rho$ in the natural way. In an index I, the substitution of the occurences of i in I by J is noted $I\{J/i\}$. We also assume that we have the subtraction as a function symbol, with $n - m = 0$ when $m \geq n$.

Definition 6 (Constraints on Indices)**.** *Let $\phi \subset \mathcal{V}$ be a 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 integers. Usually, we take $\bowtie \in \{\leq, <, =, \neq\}$. Finite set of constraints are denoted Φ.*

We say that a valuation $\rho : \mathcal{V} \to \mathbb{N}$ *satisfies* a constraint $I \bowtie J$, 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 $\phi; \Phi \vDash C$ when for all valuation ρ such that $\rho \vDash \Phi$ we have $\rho \vDash C$.

Definition 7. *The set of types and* base types *are given by the following grammar.*

$$\mathcal{B} := \mathsf{Nat}[I, J] \mid \mathsf{List}[I, J](\mathcal{B})$$

$$T := \mathcal{B} \mid \mathsf{ch}_I(\tilde{T}) \mid \mathsf{in}_I(\tilde{T}) \mid \mathsf{out}_I(\tilde{T}) \mid \forall_I \tilde{i}.\mathsf{serv}^K(\tilde{T}) \mid \forall_I \tilde{i}.\mathsf{iserv}^K(\tilde{T}) \mid \forall_I \tilde{i}.\mathsf{oserv}^K(\tilde{T})$$

A type $\mathsf{ch}_I(\tilde{T})$, $\mathsf{in}_I(\tilde{T})$ or $\mathsf{out}_I(\tilde{T})$ is called a *channel type* and a type $\forall_I \tilde{i}.\mathsf{serv}^K(\tilde{T})$, $\forall_I \tilde{i}.\mathsf{iserv}^K(\tilde{T})$ or $\forall_I \tilde{i}.\mathsf{oserv}^K(\tilde{T})$ is called a *server* type. For a channel type or a server type, the index I is called the *time* of this type, and for a server type, the index K is called the *complexity* of this type.

Intuitively, an integer n of type $\mathsf{Nat}[I, J]$ must be such that $I \leq n \leq J$. Likewise, a list of type $\mathsf{List}[I, J](\mathcal{B})$ have a size between I and J. To give a channel variable the type $\mathsf{ch}_I(\tilde{T})$ ensures that its communication should happen at time I. For example, a channel variable of type $\mathsf{ch}_0(\tilde{T})$ should do its communication before any tick occurs. Likewise, a name of type $\forall_I \tilde{i}.\mathsf{iserv}^K(\tilde{T})$ must be used in a replicated input, and this replicated input must be ready to receive for any time greater than I. Typically, a process $\mathtt{tick}.!a(\tilde{v}).P$ enforces that the type of a is $\forall_I \tilde{i}.\mathsf{iserv}^K(\tilde{T})$ with I greater than one, as the replicated input is not ready to receive at time 0.

Moreover, a server type has a kind of polymorphism for indices, and the index K stands for the complexity of the process spawned by this server. A typical example is a server taking as input a list and a channel, and send to this channel the sorted list, in time $k \cdot n$ where n is the size of the list.

$$P = !a(x, b).\overline{b}\langle \mathsf{sort}(x) \rangle$$

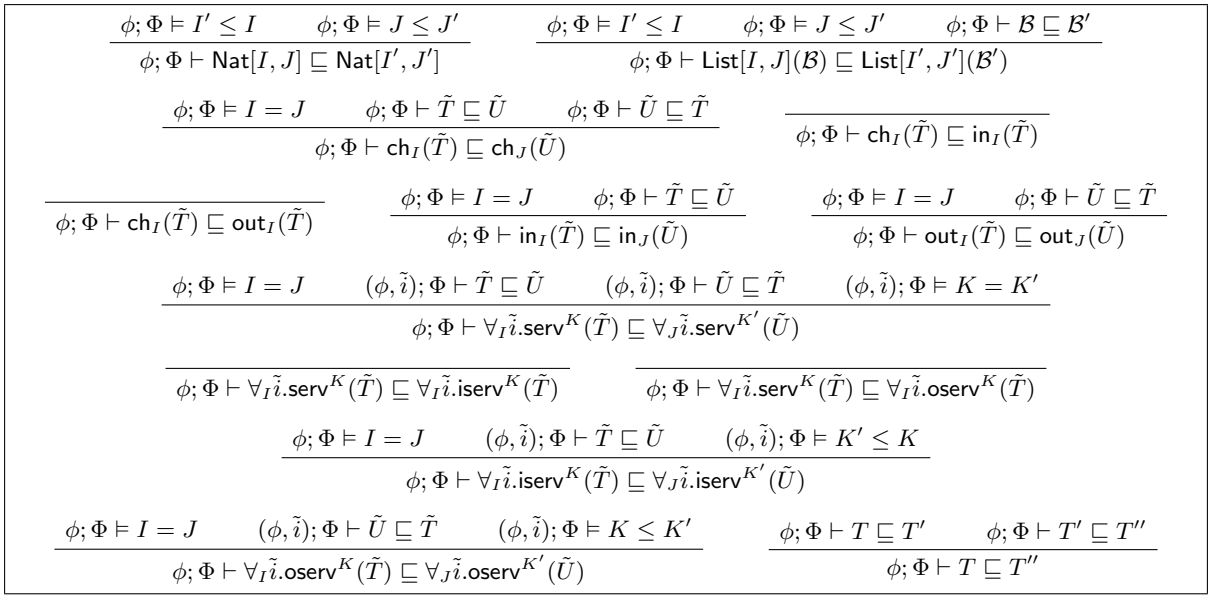

Figure 6: Subtyping Rules for Sized Types

. Such a server name a could be given the type $\forall_0 i.\mathsf{serv}^{k\cdot i}(\mathsf{List}[0,i](\mathcal{B}), \mathsf{out}_{k\cdot i}(\mathsf{List}[0,i](\mathcal{B})))$. This means that this server is ready to receive an input and, for all integer i, if given a list of size at most i and an output channel doing its communication at time $k \cdot i$ and waiting for a list of size at most i, the process spawned by this server will stop at time at most $k \cdot i$.

We define a notion of subtyping for size types. The rules are given in Figure 6.

The subtyping for channel type is standard, the only new thing is that we impose that the time of communication is invariant. Moreover, for servers, we can also change the complexity K in subtyping: for input servers, we can always define something faster than announced, and for output, we can always consider that a server is slower than announced.

In order to present the type system of our calculus, let us first introduce some notation.

Definition 8 (Advancing Time in Types)**.** *Given a set of index variables ϕ, a set of constraint Φ, a type T and an index I. We define T after I unit of time, denoted $\langle T\rangle_{-I}^{\phi;\Phi}$ by:*

- $\langle \mathcal{B}\rangle_{-I}^{\phi;\Phi} = \mathcal{B}$
- $\langle \mathsf{ch}_J(\tilde{T})\rangle_{-I}^{\phi;\Phi} = \mathsf{ch}_{(J-I)}(\tilde{T})$ *if* $\phi;\Phi \vDash J \geq I$. *It is undefined otherwise. Other channel types follow exactly the same pattern.*
- $\langle \forall_J \tilde{i}.\mathsf{serv}^K(\tilde{T})\rangle_{-I}^{\phi;\Phi} = \forall_{(J-I)}\tilde{i}.\mathsf{serv}^K(\tilde{T})$ *if* $\phi;\Phi \vDash J \geq I$. *Otherwise,* $\langle \forall_J \tilde{i}.\mathsf{serv}^K(\tilde{T})\rangle_{-I}^{\phi;\Phi} = \forall_{(J-I)}\tilde{i}.\mathsf{oserv}^K(\tilde{T})$
- $\langle \forall_J \tilde{i}.\mathsf{iserv}^K(\tilde{T})\rangle_{-I}^{\phi;\Phi} = \forall_{(J-I)}\tilde{i}.\mathsf{iserv}^K(\tilde{T})$ *if* $\phi;\Phi \vDash J \geq I$. *It is undefined otherwise.*
- $\langle \forall_J \tilde{i}.\mathsf{oserv}^K(\tilde{T})\rangle_{-I}^{\phi;\Phi} = \forall_{(J-I)}\tilde{i}.\mathsf{oserv}^K(\tilde{T})$.

This definition can be extended to contexts, with $\langle v : T, \Gamma\rangle_{-I}^{\phi;\Phi} = v : \langle T\rangle_{-I}^{\phi;\Phi}, \langle\Gamma\rangle_{-I}^{\phi;\Phi}$ *if* $\langle T\rangle_{-I}^{\phi;\Phi}$ *is defined. Otherwise,* $\langle v : T, \Gamma\rangle_{-I} = \langle\Gamma\rangle_{-I}^{\phi;\Phi}$. *We will often omit the $\phi;\Phi$ in the notation when it is clear from the context.*

Let us precise a bit the definition here. Intuitively, T after I unit of time is the type T with a time decreased by I. For base types, there is no time thus nothing happens. Then, one can wonder what happens when the time of T is smaller than I. For non-server channel types, we consider that their time is over, thus we erase them from the context. For servers this is a bit more complicated. Intuitively, when a server is defined, it should stay available until the end. Thus, an output to a server should always be possible, no matter the time. However, there are also some time limitation in servers in the sense that we must respect the time limit to define a server. As a consequence, when time advances too much,

$$\frac{v : T \in \Gamma}{\phi; \Phi; \Gamma \vdash v : T} \qquad \frac{}{\phi; \Phi; \Gamma \vdash \mathsf{0} : \mathsf{Nat}[0,0]} \qquad \frac{\phi; \Phi; \Gamma \vdash e : \mathsf{Nat}[I,J]}{\phi; \Phi; \Gamma \vdash \mathsf{s}(e) : \mathsf{Nat}[I+1, J+1]}$$

$$\frac{}{\phi; \Phi; \Gamma \vdash [] : \mathsf{List}[0,0](\mathcal{B})} \qquad \frac{\phi; \Phi; \Gamma \vdash e : \mathcal{B} \qquad \phi; \Phi; \Gamma \vdash e' : \mathsf{List}[I,J](\mathcal{B})}{\phi; \Phi; \Gamma \vdash e :: e' : \mathsf{List}[I+1, J+1](\mathcal{B})}$$

$$\frac{\phi; \Phi; \Delta \vdash e : U \qquad \phi; \Phi \vdash \Gamma \sqsubseteq \Delta \qquad \phi; \Phi \vdash U \sqsubseteq T}{\phi; \Phi; \Gamma \vdash e : T}$$

Figure 7: Typing Rules for Expressions

$$\frac{}{\phi; \Phi; \Gamma \vdash 0 \lhd 0} \qquad \frac{\phi; \Phi; \Gamma \vdash P \lhd K \qquad \phi; \Phi; \Gamma \vdash Q \lhd K}{\phi; \Phi; \Gamma \vdash P \mid Q \lhd K}$$

$$\frac{\phi; \Phi; \Gamma, \Delta \vdash a : \forall_I \tilde{i}.\mathsf{iserv}^K(\tilde{T}) \qquad (\phi, \tilde{i}); \Phi; \Gamma', \tilde{v} : \tilde{T} \vdash P \lhd K \qquad \Gamma' \text{ time invariant} \qquad \phi; \Phi \vdash \langle \Gamma \rangle_{-I}^{\phi;\Phi} \sqsubseteq \Gamma'}{\phi; \Phi; \Gamma, \Delta \vdash !a(\tilde{v}).P \lhd I}$$

$$\frac{\phi; \Phi; \Gamma \vdash a : \mathsf{in}_I(\tilde{T}) \qquad \phi; \Phi; \langle \Gamma \rangle_{-I}, \tilde{v} : \tilde{T} \vdash P \lhd K}{\phi; \Phi; \Gamma \vdash a(\tilde{v}).P \lhd K + I} \qquad \frac{\phi; \Phi; \Gamma \vdash a : \mathsf{out}_I(\tilde{T}) \qquad \phi; \Phi; \langle \Gamma \rangle_{-I} \vdash \tilde{e} : \tilde{T}}{\phi; \Phi; \Gamma \vdash \overline{a}\langle \tilde{e} \rangle \lhd I}$$

$$\frac{\phi; \Phi; \Gamma \vdash a : \forall_I \tilde{i}.\mathsf{oserv}^K(\tilde{T}) \qquad \phi; \Phi; \langle \Gamma \rangle_{-I} \vdash \tilde{e} : \tilde{T}\{\tilde{J}/\tilde{i}\}}{\phi; \Phi; \Gamma \vdash \overline{a}\langle \tilde{e} \rangle \lhd I + K\{\tilde{J}/\tilde{i}\}} \qquad \frac{\phi; \Phi; \Gamma, a : T \vdash P \lhd K}{\phi; \Phi; \Gamma \vdash (\nu a)P \lhd K}$$

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

$$\frac{\phi; \Phi; \Gamma \vdash e : \mathsf{List}[I,J](\mathcal{B}) \qquad \phi; (\Phi, I \leq 0); \Gamma \vdash P \lhd K \qquad \phi; (\Phi, J \geq 1); \Gamma, x : \mathcal{B}, y : \mathsf{List}[I-1, J-1](\mathcal{B}) \vdash Q \lhd K}{\phi; \Phi; \Gamma \vdash \mathtt{match}(e)\ \{[] \mapsto P; ;\ x :: y \mapsto Q\} \lhd K}$$

$$\frac{\phi; \Phi; \langle \Gamma \rangle_{-1} \vdash P \lhd K}{\phi; \Phi; \Gamma \vdash \mathtt{tick}.P \lhd K + 1} \qquad \frac{\phi; \Phi; \Delta \vdash P \lhd K \qquad \phi; \Phi \vdash \Gamma \sqsubseteq \Delta \qquad \phi; \Phi \vDash K \leq K'}{\phi; \Phi; \Gamma \vdash P \lhd K'}$$

Figure 8: Typing Rules for Processes

we should not be able to define a server anymore. That is why servers lose their input capability when time advances too much.

Definition 9. *Given a set of index variables ϕ and a set of constraints Φ, a context Γ is said to be* time invariant *when it contains only base type variables or output server types $\forall_I \tilde{i}.\mathsf{oserv}^K(\tilde{T})$ with $\phi; \Phi \vDash I = 0$.*

Such a context is thus invariant by the operator $\langle \cdot \rangle_{-I}$ for any I. This is typically the kind of context that we need to define a server. We can now present the type system. Rules are given in Figure 7 and Figure 8. A typing $\phi; \Phi; \Gamma \vdash P \lhd K$ means intuitively that under the constraints Φ, in the context Γ, a process P is typable and its complexity is bounded by K. And the typing for expressions $\phi; \Phi; \Gamma \vdash e : T$ means that under the constraints Φ, in the context Γ, the expression e can be given the type T.

The type system for expressions should not be very surprising. Still, remark that to type a channel name, the only possible rule is the subtyping one. In Figure 8, subtyping allows to increase the bound on the complexity. Then, the rule for parallel composition shows that we consider parallel complexity as we take the maximum between the two processes instead of the sum. In the typing for input server, we integrate some weakening on context, and we want a time invariant process to type the server, as a server should not depend on the time. Note also that a server alone has no complexity, it is a call on this server that generates complexity, as we can see in the rule for output with server types. Some rules make the time advance in their continuation, for example the tick rule or input rule. This is expressed by the advance time operator on contexts.

Finally, remark that if we remove all size annotation and merge server types and channel types together to obtain back the types of Section 2, then all the rules of Figure 7 and Figure 8 are admissible in the type system of Figure 3 and Figure 4.

Definition 10 (Forgetting Sizes)**.** *Formally, given a sized type T, we define $\mathcal{U}(T)$ the usual input/output*

type ($\mathcal{U}$ is for forgetful) by:

$$\mathcal{U}(\mathsf{Nat}[I,J]) := \mathsf{Nat} \qquad \mathcal{U}(\mathsf{List}[I,J](\mathcal{B})) := \mathsf{List}(\mathcal{U}(\mathcal{B}))$$

$$\mathcal{U}(\mathsf{ch}_I(\tilde{T})) := \mathsf{ch}(\mathcal{U}(\tilde{T})) \qquad \mathcal{U}(\mathsf{in}_I(\tilde{T})) := \mathsf{in}(\mathcal{U}(\tilde{T})) \qquad \mathcal{U}(\mathsf{out}_I(\tilde{T})) := \mathsf{out}(\mathcal{U}(\tilde{T}))$$

$$\mathcal{U}(\forall_I \tilde{i}.\mathsf{serv}^K(\tilde{T})) := \mathsf{ch}(\mathcal{U}(\tilde{T})) \qquad \mathcal{U}(\forall_I \tilde{i}.\mathsf{iserv}^K(\tilde{T})) := \mathsf{in}(\mathcal{U}(\tilde{T})) \qquad \mathcal{U}(\forall_I \tilde{i}.\mathsf{oserv}^K(\tilde{T})) := \mathsf{out}(\mathcal{U}(\tilde{T}))$$

Then, we obtain the following lemma.

Lemma 8. *If $\phi : \Phi \vdash T \sqsubseteq T'$ then $\mathcal{U}(T) \sqsubseteq \mathcal{U}(T')$. Moreover, if $\phi; \Phi; \Gamma \vdash e : T$ then $\mathcal{U}(\Gamma) \vdash e : \mathcal{U}(T)$ and if $\phi; \Phi; \Gamma \vdash P \lhd K$ then $\mathcal{U}(\Gamma) \vdash P$*

Proof. Once we have weakening for input/output types and that advancing time does not change the underlying input/output type when it is defined, the proof can be made by induction on the subtyping derivation or the typing. □

3.3 Intermediate Lemmas

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

In order to take in account the constructor $n : P$ and to work on $\Rightarrow_p$, we add another typing rule in the typing system of Figure 8.

$$\frac{\phi; \Phi; \langle \Gamma \rangle_{-n} \vdash P \lhd K}{\phi; \Phi; \Gamma \vdash n : P \lhd K + n}$$

As one can see, this rule correspond exactly to n times the rule for tick. With this, we obtain an extension of the type system for annotated processes. The goal is now to show that this extended type system gives a bound on the complexity of a annotated process, and with that, we obtain that on an usual process without this constructor ($n : Q$), the previous type system gives indeed a bound on the complexity.

Lemma 9. *If $\phi; \Phi \vdash T \sqsubseteq U$ then for any I, either $\langle U \rangle_{-I}$ is undefined, or both $\langle U \rangle_{-I}$ and $\langle T \rangle_{-I}$ are defined, and $\phi; \Phi \vdash \langle T \rangle_{-I} \sqsubseteq \langle U \rangle_{-I}$.*

Proof. We prove this by induction on the subtyping derivation. First, transitivity is direct by induction hypothesis. For non-channel type, this is direct because time advancing does not do anything. Then, for channels that are not servers, time is invariant by subtyping, thus time advancing is either undefined for both types, either defined for both types, with the same time. For a server type, again time is invariant by subtyping, so either both types lose their input capability, either they both keep the same capabilities. The case where $\langle U \rangle_{-I}$ is undefined and not $\langle T \rangle_{-I}$ is when T was an input/output server that looses its input capability, and U is an input server. □

Lemma 10 (Weakening)**.** *Let ϕ, ϕ' be disjoint set of index variables, Φ be a set of constraint on ϕ, Φ' be a set of constraints on (ϕ, ϕ'), Γ and Γ' be contexts on disjoint set of variables.*

1. *If $\phi; \Phi \vDash C$ then $(\phi, \phi'); (\Phi, \Phi') \vDash C$.*
2. *If $\phi; \Phi \vdash T \sqsubseteq U$ then $(\phi, \phi'); (\Phi, \Phi') \vdash T \sqsubseteq U$.*
3. *If $\phi; \Phi; \Gamma \vdash e : T$ then $(\phi, \phi'); (\Phi, \Phi'); \Gamma, \Gamma' \vdash e : T$.*
4. *$\langle \Gamma \rangle_{-I}^{(\phi,\phi');(\Phi,\Phi')} = \Delta, \Delta'$ with $(\phi; \phi'); (\Phi; \Phi') \vdash \Delta \sqsubseteq \langle \Gamma \rangle_{-I}^{\phi;\Phi}$.*
5. *If $\phi; \Phi; \Gamma \vdash P \lhd K$ then $(\phi, \phi'); (\Phi, \Phi'); \Gamma, \Gamma' \vdash P \lhd K$.*

Proof. Point 1 is a direct consequence of the definition of $\phi; \Phi \vDash C$. Point 2 is proved by induction on the subtyping derivation, and it uses explicitly Point 1. Point 4 is a consequence of Point 1: everything that is defined in $\langle \Gamma \rangle_{-I}^{\phi;\Phi}$ is also defined in $\langle \Gamma \rangle_{-I}^{(\phi,\phi');(\Phi,\Phi')}$, and the subtyping condition is here since with more constraints, a server may not be changed into an output server by the advance of time. Point 3 and Point 5 are proved by induction on the typing derivation, and each point uses crucially the previous ones. Note that the weakening integrated in the rule for input servers is primordial to obtain Point 5. Note also that when the advance time operator is used, the weakened typing is obtained with the use of a subtyping rule. □

We also show that we can remove some useless hypothesis.

Lemma 11 (Strengthening)**.** *Let ϕ be a set of index variables, Φ be a set of constraint on ϕ, and C a constraint on ϕ such that $\phi; \Phi \vDash C$.*

1. *If $\phi; (\Phi, C) \vDash C'$ then $\phi; \Phi \vDash C'$.*
2. *If $\phi; (\Phi, C) \vdash T \sqsubseteq U$ then $\phi; \Phi \vdash T \sqsubseteq U$.*
3. *If $\phi; (\Phi, C); \Gamma, \Gamma' \vdash e : T$ and the variables in Γ' are not free in e, then $\phi; \Phi; \Gamma \vdash e : T$.*
4. *$\langle \Gamma \rangle_{-I}^{\phi;(\Phi,C)} = \langle \Gamma \rangle_{-I}^{\phi;\Phi}$.*
5. *If $\phi; (\Phi, C); \Gamma, \Gamma' \vdash P \lhd K$ and the variables in Γ' are not free in P, then $\phi; \Phi; \Gamma \vdash P \lhd K$.*

Proof. Point 1 is a direct consequence of the definition. Point 2 is proved by induction on the subtyping derivation. Point 4 is straightforward with Point 1 of this lemma and Point 1 of Lemma 10. Point 3 and Point 5 are proved by induction on the typing derivation. □

Then, we prove that index variables can indeed be substituted by any other indexes.

Lemma 12 (Index Substitution)**.** *Let ϕ be a set of index variable and $i \notin \phi$. Let J be an index with free variables in ϕ. Then,*

1. *$[\![I\{J/i\}]\!]_\rho = [\![I]\!]_{\rho[i \mapsto [\![J]\!]_\rho]}$.*
2. *If $(\phi, i); \Phi \vDash C$ then $\phi; \Phi\{J/i\} \vDash C\{J/i\}$.*
3. *If $(\phi, i); \Phi \vdash T \sqsubseteq U$ then $\phi; \Phi\{J/i\} \vdash T\{J/i\} \sqsubseteq U\{J/i\}$.*
4. *If $(\phi, i); \Phi; \Gamma \vdash e : T$ then $\phi; \Phi\{J/i\}; \Gamma\{J/i\} \vdash e : T\{J/i\}$.*
5. *$\langle \Gamma\{J/i\} \rangle_{-I\{J/i\}}^{\phi;\Phi\{J/i\}} = \Delta, \Delta'$ with $\phi; \Phi\{J/i\} \vdash \Delta \sqsubseteq (\langle \Gamma \rangle_{-I}^{(\phi,i);\Phi})\{J/i\}$.*
6. *If $(\phi, i); \Phi; \Gamma \vdash P \lhd K$ then $\phi; \Phi\{J/i\}; \Gamma\{J/i\} \vdash P \lhd K\{J/i\}$.*

Proof. Point 1 is proved by induction on I. Then, Point 2 is a rather direct consequence of Point 1. Point 3 is proved by induction on the subtyping derivation, then Point 4 is proved by induction on the typing derivation. Point 5 is direct with the use of Point 2. And finally Point 6 is proved by induction on P. The induction is on P and not the typing derivation because of Point 5 that forces the use of weakening (Lemma 10). □

Those lemmas are rather usual in an index based system. However, the following one relies directly on our notion of time and the type system.

Lemma 13 (Delaying)**.** *Given a type T and an index I, we define the* delaying *of T by I units of time, denoted T_{+I}:*

$$\mathcal{B}_{+I} = \mathcal{B} \qquad (\mathsf{ch}_J(\tilde{T}))_{+I} = \mathsf{ch}_{J+I}(\tilde{T})$$

and for other channel and server types, the definition is the same as the one on the right above. This definition can be extended to contexts. With this, we have:

1. *If $\phi; \Phi \vdash T \sqsubseteq U$ then $\phi; \Phi \vdash T_{+I} \sqsubseteq U_{+I}$.*

2. *If* $\phi;\Phi;\Gamma \vdash e : T$ *then* $\phi;\Phi;\Gamma_{+I} \vdash e : T_{+I}$.

3. $\langle \Gamma_{+I} \rangle_{-J}^{\phi;\Phi} = \Delta, \Delta'$ *with* $\phi;\Phi \vdash \Delta \sqsubseteq (\langle \Gamma \rangle_{-J}^{\phi;\Phi})_{+I}$.

4. $\langle \Gamma_{+I} \rangle_{-(J+I)} = \langle \Gamma \rangle_{-J}$.

5. *If* $\phi;\Phi;\Gamma \vdash P \lhd K$ *then* $\phi;\Phi;\Gamma_{+I} \vdash P \lhd K + I$.

6. *For any context* Γ, $\Gamma = \Gamma', \Delta$ *with* $\phi;\Phi \vdash \Gamma' \sqsubseteq (\langle \Gamma \rangle_{-I})_{+I}$.

Proof. Point 1, Point 2, Point 3 and Point 4 are straightforward. Then, Point 5 is proved by induction on P. Point 4 is used on every rule for channel or servers, and Point 3 is used in the rule for tick. Point 6 is another straightforward proof. It is not used in the proof of Point 5 but it is useful for our subject reduction. □

We can now show the usual variable substitution lemmas.

Lemma 14 (Substitution). 1. *If* $\phi;\Phi;\Gamma, v : T \vdash e' : U$ *and* $\phi;\Phi;\Gamma \vdash e : T$ *then* $\phi;\Phi;\Gamma \vdash e'[v := e] : U$.

2. *If* $\phi;\Phi;\Gamma, v : T \vdash P \lhd K$ *and* $\phi;\Phi;\Gamma \vdash e : T$ *then* $\phi;\Phi;\Gamma \vdash P[v := e] \lhd K$.

The proof is pretty straightforward.
We can now show the subject reduction of this calculus with the reduction $\Rightarrow_p$.

3.4 Subject Reduction

The goal of this section is to prove the following theorem.

Theorem 1 (Subject Reduction). *If* $\phi;\Phi;\Gamma \vdash P \lhd K$ *and* $P \Rightarrow_p Q$ *then* $\phi;\Phi;\Gamma \vdash Q \lhd K$.

In order to do that, let us first show that the congruence relation behave well with typing.

Lemma 15 (Congruence and Typing). *Let* P *and* Q *be processes such that* $P \equiv Q$. *Then,* $\phi;\Phi;\Gamma \vdash P \lhd K$ *if and only if* $\phi;\Phi;\Gamma \vdash Q \lhd K$.

Proof. We prove this by induction on $P \equiv Q$. Remark 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 exaclty *one* subtyping rule before any syntax-directed rule. We first show this propriety for base case of congruence. The reflexivity is trivial then we have:

- **Case** $P \mid 0 \equiv P$. Suppose $\phi;\Phi;\Gamma \vdash P \mid 0 \lhd K$. Then the proof has the shape:

$$\dfrac{\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta \vdash P \lhd K'} \quad \dfrac{}{\phi;\Phi;\Delta' \vdash 0 \lhd 0} \quad \phi;\Phi \vdash \Delta \sqsubseteq \Delta'; 0 \leq K'}{\phi;\Phi;\Delta \vdash 0 \lhd K'}}{\phi;\Phi;\Delta \vdash P \mid 0 \lhd K'} \qquad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash P \mid 0 \lhd K}$$

So, we can derive the following proof:

$$\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta \vdash P \lhd K'} \qquad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash P \lhd K}$$

Reciprocally, given a proof π of $\phi;\Phi;\Gamma \vdash P \lhd K$, we can derive the proof:

$$\dfrac{\dfrac{\pi}{\phi;\Phi;\Gamma \vdash P \lhd K} \quad \dfrac{\dfrac{}{\phi;\Phi;\Gamma \vdash 0 \lhd 0} \qquad \phi;\Phi \vDash 0 \leq K}{\phi;\Phi;\Gamma \vdash 0 \lhd K}}{\phi;\Phi;\Gamma \vdash P \mid 0 \lhd K}$$

- **Case** $P \mid Q \equiv Q \mid P$. Suppose $\phi;\Phi;\Gamma \vdash P \mid Q \lhd K$. Then the proof has the shape:

$$\dfrac{\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta \vdash P \lhd K'} \quad \dfrac{\pi'}{\phi;\Phi;\Delta \vdash Q \lhd K'}}{\phi;\Phi;\Delta \vdash P \mid Q \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash P \mid Q \lhd K}$$

And so we can derive:

$$\dfrac{\dfrac{\dfrac{\pi'}{\phi;\Phi;\Delta \vdash Q \lhd K'} \quad \dfrac{\pi}{\phi;\Phi;\Delta \vdash P \lhd K'}}{\phi;\Phi;\Delta \vdash Q \mid P \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash Q \mid P \lhd K}$$

We also have the reverse in the same way.

- **Case** $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$. Suppose $\phi;\Phi;\Gamma \vdash P \mid (Q \mid R) \lhd K$. Then the proof has the shape:

$$\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta \vdash P \lhd K'} \quad \dfrac{\dfrac{\dfrac{\pi'}{\phi;\Phi;\Delta' \vdash Q \lhd K''} \quad \dfrac{\pi''}{\phi;\Phi;\Delta' \vdash R \lhd K''}}{\phi;\Phi;\Delta' \vdash Q \mid R \lhd K''} \quad \phi;\Phi \vdash \Delta \sqsubseteq \Delta'; K'' \leq K'}{\phi;\Phi;\Delta \vdash Q \mid R \lhd K'}}{\phi;\Phi;\Delta \vdash P \mid (Q \mid R) \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash P \mid (Q \mid R) \lhd K}$$

We can derive the proof:

$$\dfrac{\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta \vdash P \lhd K'} \quad \dfrac{\dfrac{\pi'}{\phi;\Phi;\Delta' \vdash Q \lhd K''} \quad \phi;\Phi \vdash \Delta \sqsubseteq \Delta'; K'' \leq K'}{\phi;\Phi;\Delta \vdash Q \lhd K'}}{\phi;\Phi;\Delta \vdash P \mid Q \lhd K'} \quad \dfrac{\dfrac{\pi''}{\phi;\Phi;\Delta' \vdash R \lhd K''} \quad \phi;\Phi \vdash \Delta \sqsubseteq \Delta'; K'' \leq K'}{\phi;\Phi;\Delta \vdash R \lhd K'}}{\phi;\Phi;\Delta \vdash (P \mid Q) \mid R \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash (P \mid Q) \mid R \lhd K}$$

The reverse follows the same pattern.

- **Case** $(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$. Suppose $\phi;\Phi;\Gamma \vdash (\nu a)(\nu b)P \lhd K$. Then the proof has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta', a : T', b : U \vdash P \lhd K''}}{\phi;\Phi;\Delta', a : T' \vdash (\nu b)P \lhd K''} \quad \phi;\Phi \vdash \Delta \sqsubseteq \Delta'; K'' \leq K'; T \sqsubseteq T'}{\phi;\Phi;\Delta, a : T \vdash (\nu b)P \lhd K'}}{\phi;\Phi;\Delta \vdash (\nu a)(\nu b)P \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash (\nu a)(\nu b)P \lhd K}$$

We can derive the proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi'}{\phi;\Phi;\Delta', a : T', b : U \vdash P \lhd K''} \quad \phi;\Phi \vdash T \sqsubseteq T'}{\phi;\Phi;\Delta', a : T, b : U \vdash P \lhd K''}}{\phi;\Phi;\Delta', b : U \vdash (\nu a)P \lhd K''}}{\phi;\Phi;\Delta' \vdash (\nu b)(\nu a)P \lhd K''} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta'; K'' \leq K}{\phi;\Phi;\Gamma \vdash (\nu b)(\nu a)P \lhd K}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta', a : T \vdash P \lhd K''}}{\phi;\Phi;\Delta' \vdash (\nu a)P \lhd K''} \quad \phi;\Phi \vdash \Delta \sqsubseteq \Delta'; K'' \leq K'}{\phi;\Phi;\Delta \vdash (\nu a)P \lhd K'} \quad \dfrac{\pi'}{\phi;\Phi;\Delta \vdash Q \lhd K'}}{\phi;\Phi;\Delta \vdash (\nu a)P \mid Q \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash (\nu a)P \mid Q \lhd K}$$

By weakening (Lemma 10), we obtain a proof π'_w of $\phi; \Phi; \Delta, a : T \vdash Q \lhd K'$. Thus, we have the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta', a : T \vdash P \lhd K''} \quad \phi;\Phi \vdash \Delta \sqsubseteq \Delta'; T \sqsubseteq T; K'' \leq K'}{\phi;\Phi;\Delta, a : T \vdash P \lhd K'} \quad \dfrac{\pi'_w}{\phi;\Phi;\Delta, a : T \vdash Q \lhd K'}}{\phi;\Phi;\Delta, a : T \vdash P \mid Q \lhd K'}}{\phi;\Phi;\Delta \vdash (\nu a)(P \mid Q) \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash (\nu a)(P \mid Q) \lhd K}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta', a : T' \vdash P \lhd K''} \quad \dfrac{\pi'}{\phi;\Phi;\Delta', a : T' \vdash Q \lhd K''}}{\phi;\Phi;\Delta', a : T' \vdash P \mid Q \lhd K''} \quad \phi;\Phi \vdash \Delta \sqsubseteq \Delta'; T \sqsubseteq T'; K'' \leq K'}{\phi;\Phi;\Delta, a : T \vdash P \mid Q \lhd K'}}{\phi;\Phi;\Delta \vdash (\nu a)(P \mid Q) \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' \leq K}{\phi;\Phi;\Gamma \vdash (\nu a)(P \mid Q) \lhd K}$$

Since a is not free in Q, by Lemma 11, from π' we obtain a proof π'_c of $\phi; \Phi; \Delta' \vdash Q \lhd K''$. We can then derive the following typing:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\phi;\Phi;\Delta', a : T' \vdash P \lhd K''} \quad \phi;\Phi \vdash T \sqsubseteq T'}{\phi;\Phi;\Delta', a : T \vdash P \lhd K''}}{\phi;\Phi;\Delta' \vdash (\nu a)P \lhd K''} \quad \dfrac{\pi'_c}{\phi;\Phi;\Delta' \vdash Q \lhd K''}}{\phi;\Phi;\Delta' \vdash (\nu a)P \mid Q \lhd K''} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta'; K'' \leq K}{\phi;\Phi;\Gamma \vdash (\nu a)P \mid Q \lhd K}$$

- **Case** $m : (P \mid Q) \equiv m : P \mid m : Q$. Suppose $\phi; \Phi; \Gamma \vdash m : (P \mid Q) \lhd K$. Then we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{\phi;\Phi;\Delta' \vdash P \lhd K''} \quad \dfrac{\pi_Q}{\phi;\Phi;\Delta' \vdash Q \lhd K''}}{\phi;\Phi;\Delta' \vdash (P \mid Q) \lhd K''} \quad \phi;\Phi \vdash \langle\Delta\rangle_{-m} \sqsubseteq \Delta'; K'' \leq K'}{\phi;\Phi;\langle\Delta\rangle_{-m} \vdash (P \mid Q) \lhd K'}}{\phi;\Phi;\Delta \vdash m : (P \mid Q) \lhd K' + m} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' + m \leq K}{\phi;\Phi;\Gamma \vdash m : (P \mid Q) \lhd K}$$

So, we can give the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{\phi;\Phi;\Delta' \vdash P \lhd K''} \quad \dfrac{\pi_Q}{\phi;\Phi;\Delta' \vdash Q \lhd K''} \quad \phi;\Phi \vdash \langle\Delta\rangle_{-m} \sqsubseteq \Delta'; K'' \leq K'}{\phi;\Phi;\langle\Delta\rangle_{-m} \vdash P \lhd K' \quad \phi;\Phi;\langle\Delta\rangle_{-m} \vdash Q \lhd K'}}{\phi;\Phi;\Delta \vdash m : P \lhd K' + m \quad \phi;\Phi;\Delta \vdash m : Q \lhd K' + m}}{\phi;\Phi;\Delta \vdash m : P \mid m : Q \lhd K' + m} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta; K' + m \leq K}{\phi;\Phi;\Gamma \vdash m : P \mid m : Q \lhd K}$$

Now, suppose we have a typing $\phi; \Phi; \Gamma \vdash m : P \mid m : Q \lhd K$. The typing has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\phi;\Phi;\langle\Gamma_P\rangle_{-m} \vdash P \lhd K_P}{\phi;\Phi;\Gamma_P \vdash m : P \lhd K_P + m} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Gamma_P; K_P + m \leq K'}{\phi;\Phi;\Delta \vdash m : P \lhd K'} \quad \dfrac{\dfrac{\phi;\Phi;\langle\Gamma_Q\rangle_{-m} \vdash Q \lhd K_P}{\phi;\Phi;\Gamma_Q \vdash m : Q \lhd K_Q + m} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Gamma_Q; K_Q + m \leq K'}{\phi;\Phi;\Gamma' \vdash m : Q \lhd K'}}{\phi;\Phi;\Gamma' \vdash m : P \mid m : Q \lhd K'} \quad \phi;\Phi \vdash \Gamma \sqsubseteq \Gamma'; K \leq K}{\phi;\Phi;\Gamma \vdash m : P \mid m : Q \lhd K}$$

By Lemma 9, we have $\Gamma \sqsubseteq \Gamma_P$ so, we have $\Gamma = \Gamma_0, \Gamma_1$ with $\langle \Gamma_1 \rangle_{-m} \sqsubseteq \langle \Gamma_P \rangle_{-m}$. In the same way, $\Gamma = \Gamma'_0, \Gamma'_1$ with $\langle \Gamma'_1 \rangle_{-m} \sqsubseteq \langle \Gamma_Q \rangle_{-m}$

By Lemma 10, we obtain:

$$\dfrac{\dfrac{\phi;\Phi;\langle\Gamma_p\rangle_{-m},\langle\Gamma_0\rangle_{-m} \vdash P \lhd K_p \quad \phi;\Phi \vdash \langle\Gamma_1\rangle_{-m} \sqsubseteq \langle\Gamma_p\rangle_{-m}; K_p \leq K' - m}{\phi;\Phi;\langle\Gamma'\rangle_{-m} \vdash P \lhd K' - m} \qquad \dfrac{\phi;\Phi;\langle\Gamma_q\rangle_{-m},\langle\Gamma'_0\rangle_{-m} \vdash Q \lhd K_q \quad \phi;\Phi \vdash \langle\Gamma\rangle_{-m} \sqsubseteq \langle\Gamma_q\rangle_{-m}; K_q \leq K' - m}{\phi;\Phi;\langle\Gamma'\rangle_{-m} \vdash Q \lhd K' - m}}{\dfrac{\phi;\Phi;\langle\Gamma'\rangle_{-m} \vdash (P \mid Q) \lhd K' - m}{\dfrac{\phi;\Phi;\Gamma' \vdash m : (P \mid Q) \lhd K' \qquad \phi;\Phi \vdash \Gamma \sqsubseteq \Gamma'; K' \leq K}{\phi;\Phi;\Gamma \vdash m : (P \mid Q) \lhd K}}}$$

This concludes this case.

- **Case** $m : (\nu a)P \equiv (\nu a)(m : P)$. To go from left to right, we only need to take a channel a with a time increased by m. To go from right to left, there are two cases to consider. Either the type T for a is suppressed after the time advance, and in this case, by weakening, we can take whatever we want for the type of a. Either $\langle T \rangle_{-m}$ is defined, and in this case we can take this type as the new type when declaring a.
- **Case** $m : (n : P) \equiv (m + n) : P$. This case is direct because $\langle T \rangle_{-(m+n)}$ is equal to $\langle \langle T \rangle_{-n} \rangle_{-m}$
- **Case** $0 : P \equiv P$. This case is direct because the rule for $0 : P$ does nothing.

This concludes all the base case. We can then prove Lemma 15 by induction on $P \equiv Q$. All the base case have been done, symmetry and transitivity are direct by induction hypothesis. For the cases of contextual congruence, the proof is again straightforward by considering proofs in which there is exaclty one subtyping rule before any syntax-directed rule. □

Now that we have Lemma 15, we can work up to the congruence relation. We now give an exhaustive description of the subtyping relation.

Lemma 16 (Exhaustive Description of Subtyping)**.** *If $\phi;\Phi \vdash T \sqsubseteq U$, then one of the following case holds.*

- $$T = \mathsf{Nat}[I, J] \qquad U = \mathsf{Nat}[I', J'] \qquad \phi;\Phi \vDash I' \leq I \qquad \phi;\Phi \vDash J \leq J'$$
- $$T = \mathsf{List}[I, J](\mathcal{B}) \qquad U = \mathsf{List}[I', J'](\mathcal{B}') \qquad \phi;\Phi \vDash I' \leq I \qquad \phi;\Phi \vDash J \leq J' \qquad \phi;\Phi \vdash \mathcal{B} \sqsubseteq \mathcal{B}'$$
- $$T = \mathsf{Bool} \qquad U = \mathsf{Bool}$$
- $$T = \mathsf{ch}_I(\tilde{T}) \qquad U = \mathsf{ch}_J(\tilde{U}) \qquad \phi;\Phi \vDash I = J \qquad \phi;\Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad \phi;\Phi \vdash \tilde{U} \sqsubseteq \tilde{T}$$
- $$T = \mathsf{ch}_I(\tilde{T}) \qquad U = \mathsf{in}_J(\tilde{U}) \qquad \phi;\Phi \vDash I = J \qquad \phi;\Phi \vdash \tilde{T} \sqsubseteq \tilde{U}$$
- $$T = \mathsf{ch}_I(\tilde{T}) \qquad U = \mathsf{out}_J(\tilde{U}) \qquad \phi;\Phi \vDash I = J \qquad \phi;\Phi \vdash \tilde{U} \sqsubseteq \tilde{T}$$
- $$T = \mathsf{in}_I(\tilde{T}) \qquad U = \mathsf{in}_J(\tilde{U}) \qquad \phi;\Phi \vDash I = J \qquad \phi;\Phi \vdash \tilde{T} \sqsubseteq \tilde{U}$$
- $$T = \mathsf{out}_I(\tilde{T}) \qquad U = \mathsf{out}_J(\tilde{U}) \qquad \phi;\Phi \vDash I = J \qquad \phi;\Phi \vdash \tilde{U} \sqsubseteq \tilde{T}$$
- $$T = \forall_I \tilde{i}.\mathsf{serv}^K(\tilde{T}) \qquad U = \forall_J \tilde{i}.\mathsf{serv}^{K'}(\tilde{U}) \qquad \phi;\Phi \vDash I = J \qquad (\phi,\tilde{i});\Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad (\phi,\tilde{i});\Phi \vdash \tilde{U} \sqsubseteq \tilde{T} \qquad (\phi,\tilde{i});\Phi \vDash K = K'$$
- $$T = \forall_I \tilde{i}.\mathsf{serv}^K(\tilde{T}) \qquad U = \forall_J \tilde{i}.\mathsf{iserv}^{K'}(\tilde{U}) \qquad \phi;\Phi \vDash I = J \qquad (\phi,\tilde{i});\Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad (\phi,\tilde{i});\Phi \vDash K' \leq K$$

- $$T = \forall_I \tilde{i}.\mathsf{serv}^K(\tilde{T}) \qquad U = \forall_J \tilde{i}.\mathsf{oserv}^{K'}(\tilde{U}) \qquad \phi; \Phi \vDash I = J \qquad (\phi, \tilde{i}); \Phi \vdash \tilde{U} \sqsubseteq \tilde{T} \qquad (\phi, \tilde{i}); \Phi \vDash K \leq K'$$
- $$T = \forall_I \tilde{i}.\mathsf{iserv}^K(\tilde{T}) \qquad U = \forall_J \tilde{i}.\mathsf{iserv}^{K'}(\tilde{U}) \qquad \phi; \Phi \vDash I = J \qquad (\phi, \tilde{i}); \Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad (\phi, \tilde{i}); \Phi \vDash K' \leq K$$
- $$T = \forall_I \tilde{i}.\mathsf{oserv}^K(\tilde{T}) \qquad U = \forall_J \tilde{i}.\mathsf{oserv}^{K'}(\tilde{U}) \qquad \phi; \Phi \vDash I = J \qquad (\phi, \tilde{i}); \Phi \vdash \tilde{U} \sqsubseteq \tilde{T} \qquad (\phi, \tilde{i}); \Phi \vDash K \leq K'$$

Proof. The proof is rather straightforward, we proceed by induction on the subtyping relation. All base cases are indeed of this form, and then for transitivity, we can use the induction hypothesis and consider all cases in which the second member of a subtyping relation can match with the first one, and all cases are simple. □

Let us now show Theorem 1. We do this by induction on $P \Rightarrow_p Q$. Let us first remark that when considering the typing of P, the first subtyping rule has no importance since we can always start the typing of Q with the exact same subtyping rule. One can see it in the detailed proof of Lemma 15. We now proceed by doing the case analysis on the rules of Figure 5.

- **Case** $(n :!a(\tilde{v}).P) \mid (m : \overline{a}\langle\tilde{e}\rangle) \Rightarrow_p (n :!a(\tilde{v}).P) \mid (max(m,n) : P[\tilde{v} := \tilde{e}])$. Consider the typing $\phi; \Phi; \Delta \vdash (n :!a(\tilde{v}).P) \mid (m : \overline{a}\langle\tilde{e}\rangle) \triangleleft K'$. The first rule is the rule for parallel composition, then the proof is split into the two following subtree:

$$\dfrac{\dfrac{\dfrac{\phi; \Phi; \Gamma_1 \vdash a : \forall_{I_1} \tilde{i}.\mathsf{oserv}^{K_1}(\tilde{T_1}) \qquad \dfrac{\pi_e}{\phi; \Phi; \langle\Gamma_1\rangle_{-I_1} \vdash \tilde{e} : \tilde{T_1}\{\tilde{J}/\tilde{i}\}}}{\phi; \Phi; \Gamma_1 \vdash \overline{a}\langle\tilde{e}\rangle \triangleleft I_1 + K_1\{\tilde{J}/\tilde{i}\}} \qquad \phi; \Phi \vdash \langle\Delta_1\rangle_{-m} \sqsubseteq \Gamma_1; I_1 + K_1\{\tilde{J}/\tilde{i}\} \leq K_1'}{\phi; \Phi; \langle\Delta_1\rangle_{-m} \vdash \overline{a}\langle\tilde{e}\rangle \triangleleft K_1'}}{\dfrac{\phi; \Phi; \Delta_1 \vdash n : \overline{a}\langle\tilde{e}\rangle \triangleleft K_1' + m \qquad \phi; \Phi \vdash \Delta \sqsubseteq \Delta_1; K_1' + m \leq K'}{\phi; \Phi; \Delta \vdash m : \overline{a}\langle\tilde{e}\rangle \triangleleft K'}}$$

$$\dfrac{\dfrac{\phi; \Phi; \Gamma_0, \Delta_0' \vdash a : \forall_{I_0} \tilde{i}.\mathsf{iserv}^{K_0}(\tilde{T_0}) \qquad \dfrac{\pi_P}{(\phi, \tilde{i}); \Phi; \Gamma', \tilde{v} : \tilde{T_0} \vdash P \triangleleft K_0} \qquad \Gamma' \text{ time invariant} \qquad \phi; \Phi \vdash \langle\Gamma_0\rangle_{-I_0} \sqsubseteq \Gamma'}{\phi; \Phi; \Gamma_0, \Delta_0' \vdash !a(\tilde{v}).P \triangleleft I_0 \qquad \phi; \Phi \vdash \langle\Delta_0\rangle_{-n} \sqsubseteq \Gamma_0, \Delta_0'; I_0 \leq K_0'}}{\dfrac{\phi; \Phi; \langle\Delta_0\rangle_{-n} \vdash !a(\tilde{v}).P \triangleleft K_0'}{\dfrac{\phi; \Phi; \Delta_0 \vdash n :!a(\tilde{v}).P \triangleleft K_0' + n \qquad \phi; \Phi \vdash \Delta \sqsubseteq \Delta_0; K_0' + n \leq K'}{\phi; \Phi; \Delta \vdash n :!a(\tilde{v}).P \triangleleft K'}}}$$

  The second subtree can be used exactly in the same way to type the server in the right part of the reduction relation. Furthermore, as the name a is used as an input and as an output, so the original type in Δ for this name must be a server type $\forall_I \tilde{i}.\mathsf{serv}^{K_a}(\tilde{T})$. As this server do not lose its input capacity after the time decrease, we also know that $\phi; \Phi \vDash I \geq n$. So, by Lemma 16, we have:

$$\phi; \Phi \vDash I_0 = I - n \qquad (\phi, \tilde{i}); \Phi \vdash \tilde{T} \sqsubseteq \tilde{T_0} \qquad (\phi, \tilde{i}); \Phi \vDash K_0 \leq K_a$$
$$\phi; \Phi \vDash I - m = I_1 \qquad (\phi, \tilde{i}); \Phi \vdash \tilde{T_1} \sqsubseteq \tilde{T} \qquad (\phi, \tilde{i}); \Phi \vDash K_a \leq K_1$$

  There are now two cases to consider.

  – Let us first consider that $\phi; \Phi \vDash I \geq m$. Then, we obtain directly:

$$\phi; \Phi \vDash I_0 + n = I_1 + m \qquad (\phi, \tilde{i}); \Phi \vdash \tilde{T_1} \sqsubseteq \tilde{T_0} \qquad (\phi, \tilde{i}); \Phi \vDash K_0 \leq K_1$$

Thus, by subtyping, from π_P we can obtain a proof of $(\phi, \tilde{i}); \Phi; \langle\Gamma_0\rangle_{-I_0}, \tilde{v} : \tilde{T_1} \vdash P \lhd K_1$. By Lemma 12, we have a proof of $\phi; \Phi\{\tilde{J}/\tilde{i}\}; (\langle\Gamma_0\rangle_{-I_0}, \tilde{v} : \tilde{T_1})\{\tilde{J}/\tilde{i}\} \vdash P \lhd K_1\{\tilde{J}/\tilde{i}\}$. As $\tilde{i}$ only appears in $\tilde{T_1}$ and K_1, we obtain a proof of $\phi; \Phi; \langle\Gamma_0\rangle_{-I_0}, \tilde{v} : \tilde{T_1}\{\tilde{J}/\tilde{i}\} \vdash P \lhd K_1\{\tilde{J}/\tilde{i}\}$.
Now, by using several times Lemma 9, we have:

$$\phi; \Phi \vdash \langle\Delta\rangle_{-(n+I_0)} \sqsubseteq \epsilon_0, \langle\Gamma_0\rangle_{-I_0} \qquad \phi; \Phi \vdash \langle\Delta\rangle_{-(m+I_1)} \sqsubseteq \epsilon_1, \langle\Gamma_1\rangle_{-I_1}$$

for some ϵ_0, ϵ_1.
By Lemma 10, and as $\phi; \Phi \vdash I = I_1 + m = I_0 + n$ we can obtain two proofs, with the subtyping rule,

$$\phi; \Phi; \langle\Delta\rangle_{-I}, \tilde{v} : \tilde{T_1}\{\tilde{J}/\tilde{i}\} \vdash P \lhd K_1\{\tilde{J}/\tilde{i}\} \qquad \phi; \Phi; \langle\Delta\rangle_{-I} \vdash \tilde{e} : \tilde{T_1}\{\tilde{J}/\tilde{i}\}$$

Thus, by the substitution lemma (Lemma 14), we have $\phi; \Phi; \langle\Delta\rangle_{-I} \vdash P[\tilde{v} := \tilde{e}] \lhd K_1\{\tilde{J}/\tilde{i}\}$. As by definition, $I \geq m$ and $I \geq n$, let us call $I' = I - max(n, m)$, and we can obtain the following typing using the associated typing rule:

$$\phi; \Phi; \langle\Delta\rangle_{-I'} \vdash max(n, m) : P[\tilde{v} := \tilde{e}] \lhd max(n, m) + K_1\{\tilde{J}/\tilde{i}\}.$$

Then, by delaying (Lemma 13), we have $\phi; \Phi; (\langle\Delta\rangle_{-I'})_{+I'} \vdash max(n, m) : P[\tilde{v} := \tilde{e}] \lhd I + K_1\{\tilde{J}/\tilde{i}\}$, and $\Delta = \epsilon_0', \epsilon_1'$ with $\phi; \Phi \vdash \epsilon_1' \sqsubseteq (\langle\Delta\rangle_{-I'})_{+I'}$. Recall that $\phi; \Phi \vDash I_1 + m + K_1\{\tilde{J}/\tilde{i}\} \leq K'$. Thus, again by subtyping and weakening, we obtain

$$\phi; \Phi; \Delta \vdash max(n, m) : P[\tilde{v} := \tilde{e}] \lhd K'$$

And this concludes this case.

– Now, we suppose that we do not have $\phi; \Phi \vdash I \geq m$. Note that as we know $\phi; \Phi \vdash I \geq n$, it means that $m > n$. Moreover, as $\phi; \Phi \vDash I_1 = I - m$, then $\phi; \Phi \vDash I_1 + m = max(I, m)$.
We still have:

$$(\phi, \tilde{i}); \Phi \vdash \tilde{T_1} \sqsubseteq \tilde{T_0} \qquad (\phi, \tilde{i}); \Phi \vDash K_0 \leq K_1$$

Thus, by subtyping, from π_P we can obtain a proof of $(\phi, \tilde{i}); \Phi; \langle\Gamma_0\rangle_{-I_0}, \tilde{v} : \tilde{T_1} \vdash P \lhd K_1$. By Lemma 12, we have a proof of $\phi; \Phi\{\tilde{J}/\tilde{i}\}; (\langle\Gamma_0\rangle_{-I_0}, \tilde{v} : \tilde{T_1})\{\tilde{J}/\tilde{i}\} \vdash P \lhd K_1\{\tilde{J}/\tilde{i}\}$. As $\tilde{i}$ only appears in $\tilde{T_1}$ and K_1, we obtain a proof of $\phi; \Phi; \Gamma', \tilde{v} : \tilde{T_1}\{\tilde{J}/\tilde{i}\} \vdash P \lhd K_1\{\tilde{J}/\tilde{i}\}$.
Now, by using several times Lemma 9, we have:

$$\phi; \Phi \vdash \langle\Delta\rangle_{-(n+I_0)} \sqsubseteq \epsilon_0, \Gamma' \qquad \phi; \Phi \vdash \langle\Delta\rangle_{-(m+I_1)} \sqsubseteq \epsilon_1, \langle\Gamma_1\rangle_{-I_1}$$

for some ϵ_0, ϵ_1.
Moreover, we know that Γ' is time invariant by definition, let us call $J = max(I_0 + n, I_1 + m)$, we have by again decreasing in the first subtyping relation:

$$\phi; \Phi \vdash \langle\Delta\rangle_{-J} \sqsubseteq \epsilon_2, \Gamma'$$

for some ϵ_2.
Note that we have $\phi; \Phi \vDash J = max(I, m) = I_1 + m$ since $\phi; \Phi \vDash I_0 + n = I$ and $\phi; \Phi \vDash I_1 + m = max(I, m)$.
By Lemma 10, we can obtain two proofs, with the subtyping rule,

$$\phi; \Phi; \langle\Delta\rangle_{-J}, \tilde{v} : \tilde{T_1}\{\tilde{J}/\tilde{i}\} \vdash P \lhd K_1\{\tilde{J}/\tilde{i}\} \qquad \phi; \Phi; \langle\Delta\rangle_{-J} \vdash \tilde{e} : \tilde{T_1}\{\tilde{J}/\tilde{i}\}$$

Thus, by the substitution lemma (Lemma 14), we have $\phi; \Phi; \langle\Delta\rangle_{-J} \vdash P[\tilde{v} := \tilde{e}] \lhd K_1\{\tilde{J}/\tilde{i}\}$. Recall that in our case, $max(m, n) = m$. We can obtain the following typing using the associated typing rule:

$$\phi; \Phi; \langle\Delta\rangle_{-(J-m)} \vdash m : P[\tilde{v} := \tilde{e}] \lhd m + K_1\{\tilde{J}/\tilde{i}\}.$$

Then, by delaying (Lemma 13), we have $\phi;\Phi;(\langle\Delta\rangle_{-(J-m)})_{+(J-m)} \vdash m : P[\tilde{v} := \tilde{e}] \lhd I_1 + m + K_1\{\tilde{J}/\tilde{i}\}$, and $\Delta = \epsilon'_0, \epsilon'_1$ with $\phi;\Phi \vdash \epsilon'_1 \sqsubseteq (\langle\Delta\rangle_{-(J-m)})_{+(J-m)}$. Recall that $\phi;\Phi \vDash I_1 + m + K_1\{\tilde{J}/\tilde{i}\} \leq K'$. Thus, again by subtyping and weakening, we obtain

$$\phi;\Phi;\Delta \vdash max(n,m) : P[\tilde{v} := \tilde{e}] \lhd K'$$

And this concludes this case. Remark that a lot of notation in this case are somewhat complicated because we only know that $\phi;\Phi \vDash I \geq m$ is false, but it does not immediately means that $\phi;\Phi \vDash m > I$ because the relation on indexes is not complete. So, we have to take that into account when writing substraction.

- **Case** $(n : a(\tilde{v}).P) \mid (m : \overline{a}\langle\tilde{e}\rangle) \Rightarrow_p max(n,m) : P[\tilde{v} := \tilde{e}]$. Consider the typing $\phi;\Phi;\Gamma \vdash (n : a(\tilde{v}).P) \mid (m : \overline{a}\langle\tilde{e}\rangle) \lhd K$. The first rule is the rule for parallel composition, then the proof is split into the two following subtree:

$$\dfrac{\dfrac{\dfrac{\phi;\Phi;\Gamma_1 \vdash a : \mathsf{out}_{I_1}(\tilde{T_1}) \qquad \dfrac{\pi_e}{\phi;\Phi;\langle\Gamma_1\rangle_{-I_1} \vdash \tilde{e} : \tilde{T_1}}}{\phi;\Phi;\Gamma_1 \vdash \overline{a}\langle\tilde{e}\rangle \lhd I_1} \qquad \phi;\Phi \vdash \langle\Delta_1\rangle_{-m} \sqsubseteq \Gamma_1; I_1 \leq K'_1}{\phi;\Phi;\langle\Delta_1\rangle_{-m} \vdash \overline{a}\langle\tilde{e}\rangle \lhd K'_1} \qquad \phi;\Phi \vdash \Delta \sqsubseteq \Delta_1; K'_1 + m \leq K'}{\phi;\Phi;\Delta_1 \vdash n : \overline{a}\langle\tilde{e}\rangle \lhd K'_1 + m}}{\phi;\Phi;\Delta \vdash m : \overline{a}\langle\tilde{e}\rangle \lhd K'}$$

$$\dfrac{\dfrac{\dfrac{\phi;\Phi;\Gamma_0 \vdash a : \mathsf{in}_{I_0}(\tilde{T_0}) \qquad \dfrac{\pi_P}{\phi;\Phi;\langle\Gamma_0\rangle_{-I_0}, \tilde{v} : \tilde{T_0} \vdash P \lhd K_0}}{\phi;\Phi;\Gamma_0 \vdash a(\tilde{v}).P \lhd I_0 + K_0} \qquad \phi;\Phi \vdash \langle\Delta_0\rangle_{-n} \sqsubseteq \Gamma_0; I_0 + K_0 \leq K'_0}{\phi;\Phi;\langle\Delta_0\rangle_{-n} \vdash a(\tilde{v}).P \lhd K'_0} \qquad \phi;\Phi \vdash \Delta \sqsubseteq \Delta_0; K'_0 + n \leq K'}{\phi;\Phi;\Delta_0 \vdash n : a(\tilde{v}).P \lhd K'_0 + n}}{\phi;\Phi;\Delta \vdash n : a(\tilde{v}).P \lhd K'}$$

As the name a is used as an input and as an output, so the original type in Δ for this name must be a channel type $\mathsf{ch}_I(\tilde{T})$. As this server do not lose its capacity after the time decrease, we also know that $\phi;\Phi \vDash I \geq n$ and $\phi;\Phi \vDash I \geq m$. So, by Lemma 16, we have:

$$\phi;\Phi \vDash I_0 = I - n \qquad (\phi,\tilde{i});\Phi \vdash \tilde{T} \sqsubseteq \tilde{T_0}$$

$$\phi;\Phi \vDash I - m = I_1 \qquad (\phi,\tilde{i});\Phi \vdash \tilde{T_1} \sqsubseteq \tilde{T}$$

So, we obtain directly:

$$\phi;\Phi \vDash I_0 + n = I_1 + m \qquad (\phi,\tilde{i});\Phi \vdash \tilde{T_1} \sqsubseteq \tilde{T_0}$$

Thus, by subtyping, from π_P we can obtain a proof of $\phi;\Phi;\langle\Gamma_0\rangle_{-I_0}, \tilde{v} : \tilde{T_1} \vdash P \lhd K_0$.

Now, by using several times Lemma 9, we have:

$$\phi;\Phi \vdash \langle\Delta\rangle_{-(n+I_0)} \sqsubseteq \epsilon_0, \langle\Gamma_0\rangle_{-I_0} \qquad \phi;\Phi \vdash \langle\Delta\rangle_{-(m+I_1)} \sqsubseteq \epsilon_1, \langle\Gamma_1\rangle_{-I_1}$$

for some ϵ_0, ϵ_1.

By Lemma 10, and as $\phi;\Phi \vdash I = I_1 + m = I_0 + n$ we can obtain two proofs, with the subtyping rule,

$$\phi;\Phi;\langle\Delta\rangle_{-I}, \tilde{v} : \tilde{T_1} \vdash P \lhd K_0 \qquad \phi;\Phi;\langle\Delta\rangle_{-I} \vdash \tilde{e} : \tilde{T_1}$$

Thus, by the substitution lemma (Lemma 14), we have $\phi;\Phi;\langle\Delta\rangle_{-I} \vdash P[\tilde{v} := \tilde{e}] \lhd K_0$. As by definition, $I \geq m$ and $I \geq n$, let us call $I' = I - max(n,m)$, and we can obtain the following typing using the associated typing rule:

$$\phi;\Phi;\langle\Delta\rangle_{-I'} \vdash max(n,m) : P[\tilde{v} := \tilde{e}] \lhd max(n,m) + K_0.$$

Then, by delaying (Lemma 13), we have $\phi; \Phi; (\langle \Delta \rangle_{-I'})_{+I'} \vdash max(n, m) : P[\tilde{v} := \tilde{e}] \lhd I + K_0$, and $\Delta = \epsilon'_0, \epsilon'_1$ with $\phi; \Phi \vdash \epsilon'_1 \sqsubseteq (\langle \Delta \rangle_{-I'})_{+I'}$. Recall that $\phi; \Phi \vDash I_0 + n + K_0 \leq K'$. Thus, again by subtyping and weakening, we obtain

$$\phi; \Phi; \Delta \vdash max(n, m) : P[\tilde{v} := \tilde{e}] \lhd K'$$

And this concludes this case.

- **Case** $\texttt{tick}.P \Rightarrow_p 1 : P$. This case id direct because both constructer have exactly the same typing rule.

- **Case** $\texttt{match}([]) \ \{[] \mapsto P; ; \ x :: y \mapsto Q\} \Rightarrow_p P$. This case is similar to its counterpart for natural number and the two case for booleans, so we only detail this one. Suppose given a derivation $\phi; \Phi; \Gamma \vdash \texttt{match}([]) \ \{[] \mapsto P; ; \ x :: y \mapsto Q\} \lhd K$. Then the derivation has the shape:

$$\dfrac{\dfrac{\dfrac{}{\phi; \Phi; \Delta \vdash [] : \mathsf{List}[0, 0](\mathcal{B}')} \qquad \phi; \Phi \vdash \Gamma \sqsubseteq \Delta; \mathsf{List}[0, 0](\mathcal{B}') \sqsubseteq \mathsf{List}[I, J](\mathcal{B})}{\phi; \Phi; \Gamma \vdash [] : \mathsf{List}[I, J](\mathcal{B})} \qquad \dfrac{\pi_P}{\phi; (\Phi, I \leq 0); \Gamma \vdash P \lhd K} \qquad \pi_Q}{\phi; \Phi; \Gamma \vdash \texttt{match}([]) \ \{[] \mapsto P; ; \ x :: y \mapsto Q\} \lhd K}$$

  Where π_Q is the typing for Q that does not interest us in this case. By Lemma 16, we obtain:

$$\phi; \Phi \vDash I \leq 0 \qquad \phi; \Phi \vDash 0 \leq J \qquad \phi; \Phi \vdash \mathcal{B}' \sqsubseteq \mathcal{B}$$

  As $\phi; \Phi \vDash I \leq 0$, by Lemma 11, we obtain directly from π_P a proof $\phi; \Phi; \Gamma \vdash P \lhd K$.

- **Case** $\texttt{match}(e :: e') \ \{[] \mapsto P; ; \ x :: y \mapsto Q\} \Rightarrow_p Q[x, y := e, e']$. This case is more difficult than its counterpart for integers, thus we only detail this case and the one for integers can easily be deduced from this one. Suppose given a derivation $\phi; \Phi; \Gamma \vdash \texttt{match}(e :: e') \ \{[] \mapsto P; ; \ x :: y \mapsto Q\} \lhd K$. Then the proof has the shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_e}{\phi; \Phi; \Delta \vdash e : \mathcal{B}'} \qquad \dfrac{\pi_{e'}}{\phi; \Phi; \Delta \vdash e' : \mathsf{List}[I', J'](\mathcal{B}')}}{\phi; \Phi; \Delta \vdash e :: e' : \mathsf{List}[I' + 1, J' + 1](\mathcal{B}')} \qquad \phi; \Phi \vdash \Gamma \sqsubseteq \Delta; \mathsf{List}[I' + 1, J' + 1](\mathcal{B}') \sqsubseteq \mathsf{List}[I, J](\mathcal{B})}{\phi; \Phi; \Gamma \vdash e :: e' : \mathsf{List}[I, J](\mathcal{B})} \qquad \pi_P \qquad \pi_Q}{\phi; \Phi; \Gamma \vdash \texttt{match}(e :: e') \ \{[] \mapsto P; ; \ x :: y \mapsto Q\} \lhd K}$$

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

  Lemma 16 gives us the following information:

$$\phi; \Phi \vDash I \leq I' + 1 \qquad \phi; \Phi \vDash J' + 1 \leq J \qquad \phi; \Phi \vdash \mathcal{B}' \sqsubseteq \mathcal{B}$$

  From this, we can deduce the following constraints:

$$\phi; \Phi \vDash J \geq 1 \qquad \phi; \Phi \vDash I - 1 \leq I' \qquad \phi; \Phi \vDash J' \leq J - 1$$

  Thus, with the subtyping rule and the proofs π_e and $\pi_{e'}$ we obtain:

$$\phi; \Phi; \Gamma \vdash e : \mathcal{B} \qquad \phi; \Phi; \Gamma \vdash e' : \mathsf{List}[I - 1, J - 1](\mathcal{B})$$

  Then, by Lemma 11, from π_Q we obtain a proof of $\phi; \Phi; \Gamma, x : \mathcal{B}, y : \mathsf{List}[I - 1, J - 1](\mathcal{B}) \vdash Q \lhd K$. By the substitution lemma (Lemma 14), we obtain $\phi; \Phi; \Gamma \vdash Q[x, y := e, e'] \lhd K$. This concludes this case.

- **Case** $P \mid R \Rightarrow_p Q \mid R$ with $P \Rightarrow_p Q$. Suppose that $\phi; \Phi; \Gamma \vdash P \mid R \lhd K$. Then the proof has the shape:

$$\dfrac{\dfrac{\pi_P}{\phi; \Phi; \Gamma \vdash P \lhd K} \qquad \dfrac{\pi_R}{\phi; \Phi; \Gamma \vdash R \lhd K}}{\phi; \Phi; \Gamma \vdash P \mid R \lhd K}$$

By induction hypothesis, with the proof π_P of $\phi; \Phi; \Gamma \vdash P \lhd K$, we obtain a proof π_Q of $\phi; \Phi; \Gamma \vdash P \lhd Q$. Then, we can derive the following proof:

$$\dfrac{\dfrac{\pi_Q}{\phi; \Phi; \Gamma \vdash Q \lhd K} \quad \dfrac{\pi_R}{\phi; \Phi; \Gamma \vdash R \lhd K}}{\phi; \Phi; \Gamma \vdash Q \mid R \lhd K}$$

This concludes this case.

- **Case** $(\nu a)P \Rightarrow_p (\nu a)Q$ with $P \Rightarrow_p Q$. Suppose that $\phi; \Phi; \Gamma \vdash (\nu a)P \lhd K$. Then the proof has the shape:

$$\dfrac{\dfrac{\pi_P}{\phi; \Phi; \Gamma, a : T \vdash P \lhd K}}{\phi; \Phi; \Gamma \vdash (\nu a)P \lhd K}$$

By induction hypothesis, with the proof π_P of $\phi; \Phi; \Gamma, a : T \vdash P \lhd K$, we obtain a proof π_Q of $\phi; \Phi; \Gamma, a : T \vdash Q \lhd K$ We can then derive the proof:

$$\dfrac{\dfrac{\pi_Q}{\phi; \Phi; \Gamma, a : T \vdash Q \lhd K}}{\phi; \Phi; \Gamma \vdash (\nu a)Q \lhd K}$$

This concludes this case.

- **Case** $n : P \Rightarrow_p n : Q$ with $P \Rightarrow_p Q$. Suppose that $\phi; \Phi; \Gamma \vdash n : P \lhd K + n$. Then, the proof has the shape:

$$\dfrac{\phi; \Phi; \langle \Gamma \rangle_{-n} \vdash P \lhd K}{\phi; \Phi; \Gamma \vdash n : P \lhd K + n}$$

By induction hypothesis, we have a proof $\phi; \Phi; \langle \Gamma \rangle_{-n} \vdash Q \lhd K$, thus we can give the derivation:

$$\dfrac{\phi; \Phi; \langle \Gamma \rangle_{-n} \vdash Q \lhd K}{\phi; \Phi; \Gamma \vdash n : Q \lhd K + n}$$

- **Case** $P \Rightarrow_p Q$ with $P \equiv P'$, $P' \Rightarrow_p Q'$ and $Q \equiv Q'$. Suppose that $\phi; \Phi; \Gamma \vdash P \lhd K$. By Lemma 15, we have $\phi; \Phi; \Gamma \vdash P' \lhd K$. By induction hypothesis, we obtain $\phi; \Phi; \Gamma \vdash Q' \lhd K$. Then, again by Lemma 15, we have $\phi; \Phi; \Gamma \vdash Q \lhd K$. This concludes this case.

This concludes the proof of Theorem 1.
We can then easily conclude the following theorem:

Theorem 2. *Let P be an annotated process and let n be its global parallel complexity. Then, if we have a typing $\phi; \Phi; \Gamma \vdash P \lhd K$, we have $\phi; \Phi \vDash K \geq n$.*

Proof. By Theorem 1, all reductions from P using $\Rightarrow_p$ conserves the typing. Moreover, for a process Q in normal form, if we have a typing $\phi; \Phi; \Gamma \vdash Q \lhd K$, then $K \geq \mathcal{C}(Q)$. Indeed, a constructor $n : P$ force a increment of the complexity of n both in typing and in the definition of $\mathcal{C}(Q)$, and for parallel composition the typing impose a complexity greater than the maximum as in the definition for $\mathcal{C}(Q)$. Thus, for all normal form Q reachable from P, we have $K \geq \mathcal{C}(Q)$, so K is indeed a bound on the parallel complexity. □

4 An example : Bitonic Sort

As an example for this type system, we show how to obtain the bound on a common parallel algorithm: bitonic sort [1]. The particularity of this sorting algorithm is that it admits a parallel complexity in $\mathcal{O}(log(n)^2)$. We will show here that our type system allows to derive this bound for the algorithm, just as the paper-and-pen analysis. Actually we consider here a version for lists, which is not optimal for the number of operations, but we obtain the usual number of comparisons. For the sake of simplicity, we present here the algorithm for lists of size a power of 2. Let us briefly sketch the ideas of this algorithm. For a formal description see [1].

- A *bitonic sequence* is either a sequence composed of an increasing sequence followed by a decreasing sequence (e.g. [2, 7, 23, 19, 8, 5]), or a cyclic rotation of such a sequence (e.g. [8, 5, 2, 7, 23, 19]).
- The algorithm uses 2 main functions, `bmerge` and `bsort`.
- `bmerge` takes a bitonic sequence and recursively sorts it, as follows:

  Assume $s = [a_0, \ldots, a_{n-1}]$ is a bitonic sequence such that $[a_0, \ldots, a_{n/2-1}]$ is increasing and $[a_{n/2} \ldots, a_{n-1}]$ is decreasing, then we consider:

  $s_1 = [\min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}) \ldots, \min(a_{n/2-1}, a_{n-1})]$

  $s_2 = [\max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}) \ldots, \max(a_{n/2-1}, a_{n-1})]$

  Then we have: s_1 and s_2 are bitonic and satisfy: $\forall x \in s_1, \forall y \in s_2, x \leq y$.

  `bmerge` then applies recursively to s_1 and s_2 to produce a sorted sequence.
- `bsort` takes a sequence and recursively sorts it. It starts by separating the sequence in two. Then, it recursively sorts the first sequence in increasing order, and the second sequence in decreasing order. With this, we obtain a bitonic sequence that can be sorted with `bmerge`.

We will encode this algorithm in π-calculus with a boolean type. As expressed before, our results can easily be extended to support boolean with a conditional constructor.

First, we suppose that a server for comparison `lessthan` is already implemented. We start with `bcompare` such that given two lists of same length, it creates the list of maximum and the list of minimum. This is described in Figure 9.

We present here intuitively the typing. A sketch of the formal typing is given in Figure 10. To begin with, we suppose that `lessthan` is given the server type ${}_0\mathsf{oserv}^0(\mathcal{B}, \mathcal{B}, \mathsf{ch}_0(\mathsf{Bool}))$, saying that this is a server ready to be called, and it takes in input a channel that is used to return the boolean value. With this, we can give to `bcompare` the following server type:

$$\forall_0 i.\mathsf{serv}^1(\mathsf{List}[0,i](\mathcal{B}), \mathsf{List}[0,i](\mathcal{B}), \mathsf{out}_1(\mathsf{List}[0,i](\mathcal{B}), \mathsf{List}[0,i](\mathcal{B})))$$

The important things to notice is that this server has complexity 1, and the channel taken in input has a time 1. In order to verify that this type is correct, we would first need to apply the rule for replicated input. Let us denote by Γ the hypothesis on those two servers names, and Γ' be as Γ except that for `bcompare` we only have the output capability. Then, Γ' is indeed time invariant, and we have $\vdash \langle\Gamma\rangle_{-0} \sqsubseteq \Gamma'$, so we can continue the typing with this context Γ'. Then, we need to show that the process after the replicated input indeed has complexity 1. In the cases of empty list, this can be done easily. In the non-empty case, for the ν constructor, we must give a type to the channels b and c. We use:

$$b : \mathsf{ch}_1(\mathsf{List}[0, i{-}1](\mathcal{B}), \mathsf{List}[0, i{-}1](\mathcal{B})) \qquad c : \mathsf{ch}_1(\mathsf{Bool})$$

And we can then type the different processes in parallel.

- For the call to `bcompare`, the arguments have the expected type, and this call has complexity 1 because of the type of `bcompare`.
- For the process $\mathtt{tick}.\overline{\mathtt{lessthan}}\langle x, y, c\rangle$, the tick enforces a decreasing of time 1 in the context. This modifies in particular the time of c, that becomes 0. Thus, we can do the call to `lessthan` as everything is well-typed.

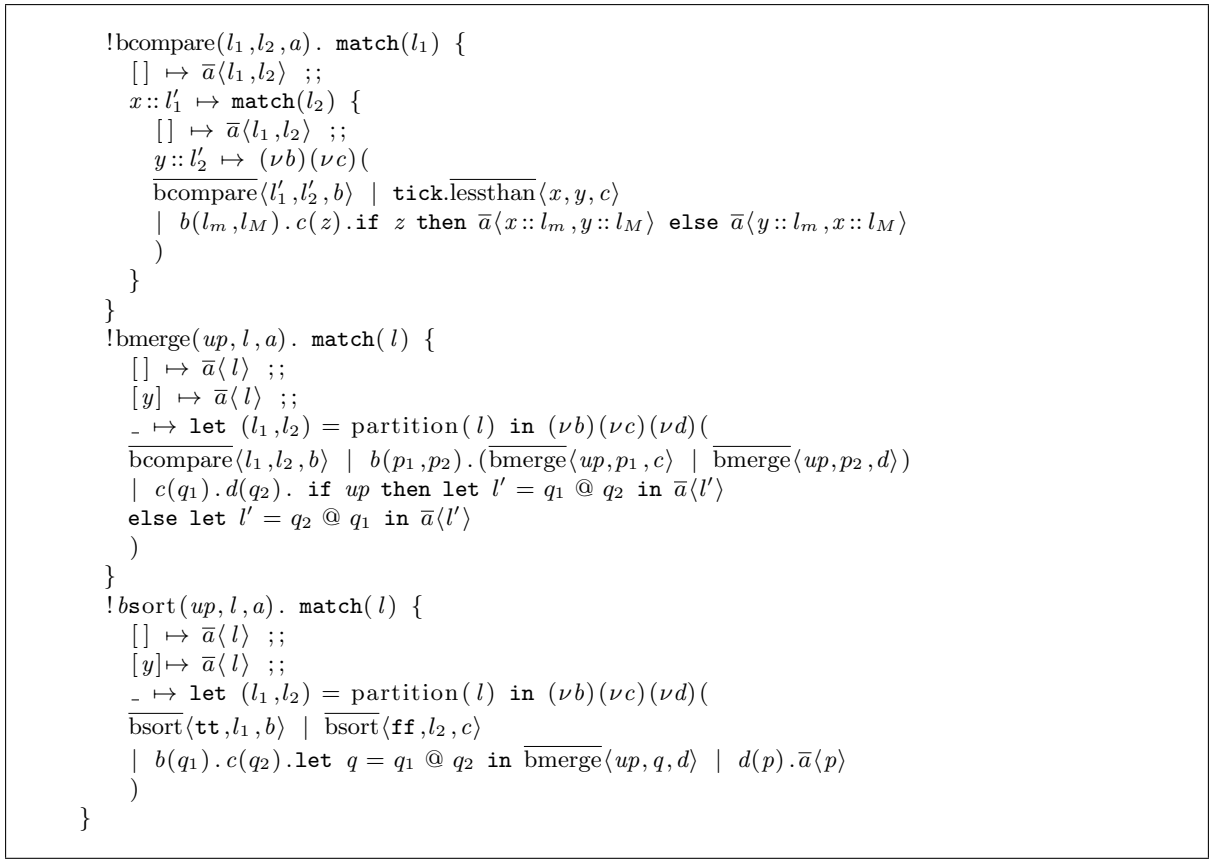

Figure 9: Bitonic Sort

- Finally, for the last process, because b has a time equal to 1, the first input has complexity 1 and it enforces again a decreasing of 1 time unit. In particular, the times of c and a become 0. Then, as there is no more `tick` and all channels have time 0, the typing proceeds without difficulties.

So, we can indeed give this server type to `bcompare`, and thus we can call this server and it generates a complexity of 1.

Then, to present the process for bitonic sort, let us use the macro `let` $\tilde{v} = f(\tilde{e})$ `in` P to represent $(\nu a)(\overline{f}\langle \tilde{e}, a\rangle \mid a(\tilde{v}).P)$, and let us also use a generalized pattern matching. We also assume that we have a function for concatenation of lists and a function `partition` taking a list of size $2n$, and giving two lists corresponding to the first n elements and the last n elements. Then, the process for bitonic sort is given in Figure 9.

Without going into details, the main point in the typing of those relations is to find a solution to a recurrence relation for the complexity of server types. In the typing of `bmerge`, we suppose given a list of size smaller than 2^i and we choose both the complexity of this type and the time of the channel a equal to a certain index K (with i free in K). So, it means we chose for `bmerge` the type:

$$\forall_0 i.\mathsf{serv}^K(\mathsf{Bool}, \mathsf{List}[0, 2^i](\mathcal{B}), \mathsf{out}_K(\mathsf{List}[0, 2^i](\mathcal{B})))$$

Then, the typing gives us the following condition.

$$i \geq 1 \text{ implies } K \geq 1 + K\{i\text{–}1/i\}$$

Indeed, the two recursive calls to `bmerge` are done after one unit of time (because the input $b(p_1, p_2)$ takes one unit of time, as expressed by the type of `bcompare`), and with a list of size 2^{i-1}. And then, the continuation after those recursive calls (the process after $c(q_1).d(q_2)$) does not generate any complexity. So, we can take $K = i$, and thus `bmerge` has logarithmic complexity. Then, in the same way we obtain a recurrence relation for the complexity K' of `bsort` on an input list of size smaller than 2^i.

$$i \geq 1 \text{ implies } K' \geq K'\{i\text{–}1/i\} + i$$

$$\dfrac{\dfrac{i;i\geq 1;\langle\Delta'\rangle_{-0}\vdash l_1',l_2',b:\tilde{T}\{i{-}1/i\}}{i;i\geq 1;\Delta'\vdash \overline{bcompare}\langle l_1',l_2',b\rangle\lhd 1}\quad \dfrac{\dfrac{i;i\geq 1;\langle\Delta'\rangle_{-1}\vdash c:\mathsf{ch}_0(\mathsf{Bool})}{i;i\geq 1;\langle\Delta'\rangle_{-1}\vdash \overline{lessthan}\langle y,y',c\rangle\lhd 0}}{i;i\geq 1;\Delta'\vdash \texttt{tick}.\overline{lessthan}\langle y,y',c\rangle\lhd 1}\quad \dfrac{}{i;i\geq 1;\Delta''\vdash \texttt{if}\ \cdots\lhd 0}}{i;i\geq 1;\Delta'\vdash b(l_{min},l_{max}).c(z).\texttt{if}\ \cdots\lhd 1}$$

$$\dfrac{i;i\geq 1;\Delta'\vdash \overline{bcompare}\langle l_1',l_2',b\rangle\mid \texttt{tick}.\overline{lessthan}\langle y,y',c\rangle\mid b(l_{min},l_{max}).c(z).\texttt{if}\ \cdots\lhd 1}{\dfrac{i;\cdot;\Delta\vdash \texttt{match}\ l_1\ \texttt{with}\ \cdots\lhd 1\qquad \Gamma'\text{ time invariant}\qquad \cdot;\cdot\vdash\langle\Gamma\rangle_{-0}\sqsubseteq\Gamma'}{\cdot;\cdot;\Gamma\vdash !bcompare(l_1,l_2,a).\ldots\lhd 0}}$$

with $\tilde{T}=\mathsf{List}[0,i](\mathcal{B}),\mathsf{List}[0,i](\mathcal{B}),\mathsf{out}_1(\mathsf{List}[0,i](\mathcal{B}),\mathsf{List}[0,i](\mathcal{B}))$
$\Gamma = lessthan:\ {}_0\mathsf{oserv}^0(\mathcal{B},\mathcal{B},\mathsf{ch}_0(\mathsf{Bool})), bcompare:\forall_0 i.\mathsf{serv}^1(\tilde{T})$
$\Gamma' = lessthan:\ {}_0\mathsf{oserv}^0(\mathcal{B},\mathcal{B},\mathsf{ch}_0(\mathsf{Bool})), bcompare:\forall_0 i.\mathsf{oserv}^1(\tilde{T})$
$\Delta=\Gamma',(l_1,l_2,a):\tilde{T}$
$\Delta'=\Delta,y:\mathcal{B},l_1':\mathsf{List}[0,i{-}1](\mathcal{B}),y':\mathcal{B},l_2':\mathsf{List}[0,i{-}1](\mathcal{B}),b:\mathsf{ch}_1(\mathsf{List}[0,i{-}1](\mathcal{B}),\mathsf{List}[0,i{-}1](\mathcal{B})),c:\mathsf{ch}_1(\mathsf{Bool})$
$\Delta''=\langle\Delta'\rangle_{-1},l_{min}:\mathsf{List}[0,i{-}1](\mathcal{B}),l_{max}:\mathsf{List}[0,i{-}1](\mathcal{B}),z:\mathsf{Bool}$

Figure 10: A sketch of type derivation for bitonic comparison

$$\dfrac{P\to_1 P'}{P\mid Q\to_1 P'\mid Q}\qquad \dfrac{Q\to_1 Q'}{P\mid Q\to_1 P\mid Q}\qquad \dfrac{P\to_1 P'}{(\nu a)P\to_1(\nu a)P'}\qquad \dfrac{}{\texttt{tick}.P\to_1 P}$$

Figure 11: Tick Reduction Rules

Again, the two recursive calls are done on lists of size 2^{i-1}. This time, the delay of i in the recurrence relation is given by the continuation, because of the call to `bmerge` that generates a complexity of i. Thus, we can take a K' in $\mathcal{O}(i^2)$, and we obtain in the end that bitonic sort is indeed in $\mathcal{O}(log(n)^2)$ on a list of size n.

Remark that in this example, the type system gives recurrence relations corresponding to the usual recurrence relations we would obtain with a complexity analysis by hand. Here, the recurrence relation is only on K because channel names are only used as return channels, so their time is always equal to the complexity of the server that uses them. In general this is not the case, so we obtain mutually recurrent relations when defining a server.

5 Work of a Process

We now want to obtain the total complexity of a process, that is to say the total number of tick without parallelism. We will see that this notion of complexity is far easier to obtain. First, let us define the new time reduction we are interested in $\to_1$. This is defined in Figure 11.

And then, from any process P, a reduction to Q is just a sequence of one-step reductions with $\to$ or $\to_1$, and the complexity of this reduction is the number of $\to_1$. We will now again design a type system to obtain a bound on the complexity of all possible reductions from P. We will see that this type system is more permissive than the previous one, and is a simplification of the previous one.

Definition 11. *The set of types and* base types *are given by the following grammar.*

$$\mathcal{B} := \mathsf{Nat}[I,J] \mid \mathsf{List}[I,J](\mathcal{B})$$
$$T := \mathcal{B} \mid \mathsf{ch}(\tilde{T}) \mid \mathsf{in}(\tilde{T}) \mid \mathsf{out}(\tilde{T}) \mid \forall\tilde{i}.\mathsf{serv}^K(\tilde{T}) \mid \forall\tilde{i}.\mathsf{iserv}^K(\tilde{T}) \mid \forall\tilde{i}.\mathsf{oserv}^K(\tilde{T})$$

Note that there are no time indication in those types. Then, the subtyping system is given in Figure 12. It is very close to the previous one.

And then, the typing for expressions is the same as before, and for processes we take the rules of Figure 13.

With this type system, we obtain as before some lemmas such as weakening (Lemma 10), strengthening (Lemma 11), index substitution (Lemma 12) and finally substitution (Lemma 14). With those, we

$$\frac{\phi;\Phi \vDash I' \leq I \qquad \phi;\Phi \vDash J \leq J'}{\phi;\Phi \vdash \mathsf{Nat}[I,J] \sqsubseteq \mathsf{Nat}[I',J']} \qquad \frac{\phi;\Phi \vDash I' \leq I \qquad \phi;\Phi \vDash J \leq J' \qquad \phi;\Phi \vdash \mathcal{B} \sqsubseteq \mathcal{B}'}{\phi;\Phi \vdash \mathsf{List}[I,J](\mathcal{B}) \sqsubseteq \mathsf{List}[I',J'](\mathcal{B}')}$$

$$\frac{\phi;\Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad \phi;\Phi \vdash \tilde{U} \sqsubseteq \tilde{T}}{\phi;\Phi \vdash \mathsf{ch}(\tilde{T}) \sqsubseteq \mathsf{ch}(\tilde{U})} \qquad \frac{}{\phi;\Phi \vdash \mathsf{ch}(\tilde{T}) \sqsubseteq \mathsf{in}(\tilde{T})}$$

$$\frac{}{\phi;\Phi \vdash \mathsf{ch}(\tilde{T}) \sqsubseteq \mathsf{out}(\tilde{T})} \qquad \frac{\phi;\Phi \vdash \tilde{T} \sqsubseteq \tilde{U}}{\phi;\Phi \vdash \mathsf{in}(\tilde{T}) \sqsubseteq \mathsf{in}(\tilde{U})} \qquad \frac{\phi;\Phi \vdash \tilde{U} \sqsubseteq \tilde{T}}{\phi;\Phi \vdash \mathsf{out}(\tilde{T}) \sqsubseteq \mathsf{out}(\tilde{U})}$$

$$\frac{(\phi,\tilde{i});\Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad (\phi,\tilde{i});\Phi \vdash \tilde{U} \sqsubseteq \tilde{T} \qquad (\phi,\tilde{i});\Phi \vDash K = K'}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{serv}^K(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{serv}^{K'}(\tilde{U})}$$

$$\frac{}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{serv}^K(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{iserv}^K(\tilde{T})} \qquad \frac{}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{serv}^K(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{oserv}^K(\tilde{T})}$$

$$\frac{(\phi,\tilde{i});\Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad (\phi,\tilde{i});\Phi \vDash K' \leq K}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{iserv}^K(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{iserv}^{K'}(\tilde{U})}$$

$$\frac{(\phi,\tilde{i});\Phi \vdash \tilde{U} \sqsubseteq \tilde{T} \qquad (\phi,\tilde{i});\Phi \vDash K \leq K'}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{oserv}^K(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{oserv}^{K'}(\tilde{U})} \qquad \frac{\phi;\Phi \vdash T \sqsubseteq T' \qquad \phi;\Phi \vdash T' \sqsubseteq T''}{\phi;\Phi \vdash T \sqsubseteq T''}$$

Figure 12: Subtyping Rules for Sized Types

$$\frac{}{\phi;\Phi;\Gamma \vdash 0 \lhd 0} \qquad \frac{\phi;\Phi;\Gamma \vdash P \lhd K \qquad \phi;\Phi;\Gamma \vdash Q \lhd K'}{\phi;\Phi;\Gamma \vdash P \mid Q \lhd K + K'}$$

$$\frac{\phi;\Phi;\Gamma \vdash a : \forall\tilde{i}.\mathsf{iserv}^K(\tilde{T}) \qquad (\phi,\tilde{i});\Phi;\Gamma,\tilde{v} : \tilde{T} \vdash P \lhd K}{\phi;\Phi;\Gamma \vdash !a(\tilde{v}).P \lhd 0}$$

$$\frac{\phi;\Phi;\Gamma \vdash a : \mathsf{in}(\tilde{T}) \qquad \phi;\Phi;\Gamma,\tilde{v} : \tilde{T} \vdash P \lhd K}{\phi;\Phi;\Gamma \vdash a(\tilde{v}).P \lhd K} \qquad \frac{\phi;\Phi;\Gamma \vdash a : \mathsf{out}(\tilde{T}) \qquad \phi;\Phi;\Gamma \vdash \tilde{e} : \tilde{T}}{\phi;\Phi;\Gamma \vdash \overline{a}\langle\tilde{e}\rangle \lhd 0}$$

$$\frac{\phi;\Phi;\Gamma \vdash a : \forall\tilde{i}.\mathsf{oserv}^K(\tilde{T}) \qquad \phi;\Phi;\Gamma \vdash \tilde{e} : \tilde{T}\{\tilde{J}/\tilde{i}\}}{\phi;\Phi;\Gamma \vdash \overline{a}\langle\tilde{e}\rangle \lhd K\{\tilde{J}/\tilde{i}\}} \qquad \frac{\phi;\Phi;\Gamma,a : T \vdash P \lhd K}{\phi;\Phi;\Gamma \vdash (\nu a)P \lhd K}$$

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

$$\frac{\phi;\Phi;\Gamma \vdash e : \mathsf{List}[I,J](\mathcal{B}) \qquad \phi;(\Phi, I \leq 0);\Gamma \vdash P \lhd K \qquad \phi;(\Phi, J \geq 1);\Gamma, x : \mathcal{B}, y : \mathsf{List}[I-1,J-1](\mathcal{B}) \vdash Q \lhd K}{\phi;\Phi;\Gamma \vdash \mathtt{match}(e)\ \{[] \mapsto P;;\ x :: y \mapsto Q\} \lhd K}$$

$$\frac{\phi;\Phi;\Gamma \vdash P \lhd K}{\phi;\Phi;\Gamma \vdash \mathtt{tick}.P \lhd K + 1} \qquad \frac{\phi;\Phi;\Delta \vdash P \lhd K \qquad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta \qquad \phi;\Phi \vDash K \leq K'}{\phi;\Phi;\Gamma \vdash P \lhd K'}$$

Figure 13: Typing Rules for Processes

can show with a simpler proof than before the non-quantitative subject reduction (Theorem 1). In this case, it is written:

Theorem 3 (Non Quantitative Subject Reduction for Work)**.** *If* $\phi; \Phi; \Gamma \vdash P \lhd K$ *and* $P \to Q$ *then* $\phi; \Phi; \Gamma \vdash Q \lhd K$*.*

We do not detail this proof as this is really the same proof as Theorem 1, but without all the difficulties with time and the constructor $n : P$.

Then, we can show the following theorem:

Theorem 4 (Work Complexity)**.** *If* $P \to_1 Q$ *and* $\phi; \Phi; \Gamma \vdash P \lhd K$ *then we have* $\phi; \Phi; \Gamma \vdash Q \lhd K'$ *with* $\phi; \Phi \vDash K' + 1 \leq K$*.*

Proof. By induction on $P \to_1 Q$. All the cases are direct, since the rule for parallel composition is the sum of complexity and the rule for ν does not change the complexity. Finally, the rule for tick gives directly this propriety. □

So, as a consequence we obtain quasi immediately that K is indeed a bound on the complexity of P if we have $\phi; \Phi; \Gamma \vdash P \lhd K$. As we can see, this complexity is far more easier to obtain than the span as the parallelism is not really taken in account.

6 A Hint for Type Inference

We describe in this section how could we design type inference for those kind of type systems. We illustrate here on the type system for work as it is easier, but we believe we could extend the reasoning for span, as time should not be too different from the complexity of a server type, and the operation $\langle \Gamma \rangle_{-I}$ should be manageable when reading from top to bottom. The main idea is to consider a subsystem of our current type system in order to have something very close to a known inferrable type system [2]. In this paper, the main point is the use of canonical form for a type system with size for functional programs. Intuitively, this canonical form forces the types of an input function to have the shape $\mathsf{Nat}[0, i, J]$ for some new fresh variable i. In a type, the only places where we can have more complex indices (such as $i^2 + j$ or other expressions) is at positions that correspond to outputs.

So, what we will do is to give a type system inspired with this canonical form. Then, if we can show that this type systme is a restruction of the previous one for work, we obtain automatically the soundness of this type system (with regard to work complexity). We can then use a similar inference procedure as [2]. We will not detail this procedure here, as it is largely inspired by this paper. The procedure generates constraints that must be satisfied in order to type a process, and then this set of constraints could be, for example, given to a SMT solver.

So, there are two steps to this type inference, first we need to be able to generates constraints from a process such that if this set of constraints is satisfiable, then the initial process was typable. In order to show expressivity, we would also like that if a process is typable, then the generated set of constraints is indeed satisfiable. With this, we know that we do not lose expressivity with the reduction to a constraint satisfaction problem. Then, the second step, is to give those constraints to a SMT solver and hope that it can solve it (recall that this is undecidable in practice).

In this section, we will only describe this intermediate type system with canonical form inspired from [2], and we do not describe the full procedure as it is not so interesting, it really looks like the one of this paper. Still, note that we have a proof that this procedure is sound and complete for this intermediate type system.

First, we change the behaviour of channels, in order to have a quantification for channel variable even for channel types, then we merge server and channel together.

Definition 12. *The set of types and* base types *are given by the following grammar.*

$$\mathcal{B} := \mathsf{Nat}[I, J] \mid \mathsf{List}[I, J](\mathcal{B}) \mid \mathsf{Bool}$$

$$T := \mathcal{B} \mid \forall \tilde{i}.\mathsf{ch}^K(\tilde{T}) \mid \forall \tilde{i}.\mathsf{in}^K(\tilde{T}) \mid \forall \tilde{i}.\mathsf{out}^K(\tilde{T})$$

$$\frac{\phi;\Phi \vDash I' \leq I \qquad \phi;\Phi \vDash J \leq J'}{\phi;\Phi \vdash \mathsf{Nat}[I,J] \sqsubseteq \mathsf{Nat}[I',J']} \qquad \frac{\phi;\Phi \vDash I' \leq I \qquad \phi;\Phi \vDash J \leq J' \qquad \phi;\Phi \vdash \mathcal{B} \sqsubseteq \mathcal{B}'}{\phi;\Phi \vdash \mathsf{List}[I,J](\mathcal{B}) \sqsubseteq \mathsf{List}[I',J'](\mathcal{B}')}$$

$$\frac{}{\phi;\Phi \vdash \mathsf{Bool} \sqsubseteq \mathsf{Bool}} \qquad \frac{(\phi,\tilde{i});\Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad (\phi,\tilde{i});\Phi \vdash \tilde{U} \sqsubseteq \tilde{T} \qquad (\phi,\tilde{i});\Phi \vDash K = K'}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{ch}^{K}(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{ch}^{K'}(\tilde{U})}$$

$$\frac{}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{ch}^{K}(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{in}^{K}(\tilde{T})} \qquad \frac{}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{ch}^{K}(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{out}^{K}(\tilde{T})}$$

$$\frac{(\phi,\tilde{i});\Phi \vdash \tilde{T} \sqsubseteq \tilde{U} \qquad (\phi,\tilde{i});\Phi \vDash K' \leq K}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{in}^{K}(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{in}^{K'}(\tilde{U})} \qquad \frac{(\phi,\tilde{i});\Phi \vdash \tilde{U} \sqsubseteq \tilde{T} \qquad (\phi,\tilde{i});\Phi \vDash K \leq K'}{\phi;\Phi \vdash \forall\tilde{i}.\mathsf{out}^{K}(\tilde{T}) \sqsubseteq \forall\tilde{i}.\mathsf{out}^{K'}(\tilde{U})}$$

$$\frac{\phi;\Phi \vdash T \sqsubseteq T' \qquad \phi;\Phi \vdash T' \sqsubseteq T''}{\phi;\Phi \vdash T \sqsubseteq T''}$$

Figure 14: Subtyping Rules for Sized Types, for Work Inference

$$\frac{}{\phi;\Phi;\Gamma \vdash 0 \lhd 0} \qquad \frac{\phi;\Phi;\Gamma \vdash P \lhd K \qquad \phi;\Phi;\Gamma \vdash Q \lhd K'}{\phi;\Phi;\Gamma \vdash P \mid Q \lhd K + K'}$$

$$\frac{\phi;\Phi;\Gamma, a : T \vdash P \lhd K}{\phi;\Phi;\Gamma \vdash (\nu a)P \lhd K} \qquad \frac{\phi;\Phi;\Gamma \vdash a : \forall\tilde{i}.\mathsf{in}^{K}(\tilde{T}) \qquad (\phi,\tilde{i});\Phi;\Gamma,\tilde{v} : \tilde{T} \vdash P \lhd K}{\phi;\Phi;\Gamma \vdash !a(\tilde{v}).P \lhd 0}$$

$$\frac{\phi;\Phi;\Gamma \vdash a : \forall\tilde{i}.\mathsf{in}^{K}(\tilde{T}) \qquad (\phi,\tilde{i});\Phi;\Gamma,\tilde{v} : \tilde{T} \vdash P \lhd K}{\phi;\Phi;\Gamma \vdash a(\tilde{v}).P \lhd 0} \qquad \frac{\phi;\Phi;\Gamma \vdash a : \forall\tilde{i}.\mathsf{out}^{K}(\tilde{T}) \qquad \phi;\Phi;\Gamma \vdash \tilde{e} : \tilde{T}\{\tilde{J}/\tilde{i}\}}{\phi;\Phi;\Gamma \vdash \overline{a}\langle\tilde{e}\rangle \lhd K\{\tilde{J}/\tilde{i}\}}$$

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

$$\frac{\phi;\Phi;\Gamma \vdash e : \mathsf{List}[I,J](\mathcal{B}) \qquad \phi;(\Phi, I \leq 0);\Gamma \vdash P \lhd K \qquad \phi;(\Phi, J \geq 1);\Gamma, x : \mathcal{B}, y : \mathsf{List}[I-1,J-1](\mathcal{B}) \vdash Q \lhd K}{\phi;\Phi;\Gamma \vdash \mathtt{match}(e)\ \{[] \mapsto P;;\ x :: y \mapsto Q\} \lhd K}$$

$$\frac{\phi;\Phi;\Gamma \vdash e : \mathsf{Bool} \qquad \phi;\Phi;\Gamma \vdash P \lhd K \qquad \phi;\Phi;\Gamma \vdash Q \lhd K}{\phi;\Phi;\Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \lhd K} \qquad \frac{\phi;\Phi;\Gamma \vdash P \lhd K}{\phi;\Phi;\Gamma \vdash \mathtt{tick}.P \lhd K + 1}$$

$$\frac{\phi;\Phi;\Delta \vdash P \lhd K \qquad \phi;\Phi \vdash \Gamma \sqsubseteq \Delta \qquad \phi;\Phi \vDash K \leq K'}{\phi;\Phi;\Gamma \vdash P \lhd K'}$$

Figure 15: Typing Rules for Processes for Work Inference

With the associated type system described in Figure 14 and 15. Note that the only modification is that channel types and server types are not distinct anymore, however, since for the work there is no real differences between them, this do not change the proof for the complexity bound. Note that this operation for span is not so simple, as server have an important distinction from channel when time advances.

6.1 The Restricted Type System

In order to obtain a procedure for our type system, we would first need to begin by a classical algorithm such that, given a process P, this algorithm gives a classical π-calculus type to this process P. This is a standard type inference algorithm. This algorithm output a type in the following grammar:

$$\mathcal{B} := \mathsf{Nat} \mid \mathsf{List}(\mathcal{B}) \mid \mathsf{Bool} \mid \alpha, \beta, \ldots$$

$$T := \mathcal{B} \mid \mathsf{ch}(\tilde{T}) \mid A, B, \ldots$$

Where $\alpha, \beta, \ldots$ are variables for base types, and $A, B, \ldots$ are variables for channel types. We may use also variables for base types and channel types when we can to obtain a general shape. We will not detail it here, as the theory is very similar to [2], However, our procedure will not be complete, in the

sense that there will be typable program for which we may not obtain a satisfiable set of constraints. This is because we will restraints the possible types, thus some process may not be typable anymore. In order to still show some expressivity in our procedure, we present an intermediate type system such that being typable for this type system implies being typable for the work type system, and we show that our procedure is sound and complete for this new type system.

Intermediate Type System. For this intermediate type system, we consider the following grammar.

$$\mathcal{B} := \mathsf{Nat}[0, I] \mid \mathsf{List}[0, I](\mathcal{B}) \mid \mathsf{Bool} \mid \alpha, \beta, \dots$$
$$T := \mathcal{B} \mid \forall \tilde{i}.\mathsf{ch}^K(\tilde{T}) \mid A, B, \dots$$

So this is typically the usual type system without input/output, and with types variables (useful for type inference). Then, we consider types in the following canonical form:

Definition 13. *Canonical Intermediate Types* A canonical intermediate type *is a type given by the following grammar:*

$$\mathcal{B}_c := \mathsf{Nat}[0, i+n] \mid \mathsf{List}[0, i+n](\mathcal{B}_c) \mid \mathsf{Bool} \mid \alpha, \beta, \dots$$
$$T_c := \mathcal{B}_c \mid \forall i_1, \dots, i_m.\mathsf{ch}^K(\tilde{T_c}) \mid A, B, \dots$$

Where i is an index variable, n an integer and in the channel type, the index variables of K are in $\{i_1, \dots, i_m\}$ and m is equal to the number of base type index occurrences in $\tilde{T_c}$, noted $\mathtt{btocc}(\tilde{T_c})$, and defined by:

- $\mathtt{btocc}(T_c^1, \cdots T_c^k) = \mathtt{btocc}(T_c^1) + \cdots + \mathtt{btocc}(T_c^k)$
- $\mathtt{btocc}(\mathsf{Nat}[0, i]) = 1$
- $\mathtt{btocc}(\mathsf{List}[0, i](\mathcal{B}_c)) = 1 + \mathtt{btocc}(\mathcal{B}_c)$
- $\mathtt{btocc}(\mathsf{Bool}) = \mathtt{btocc}(\alpha) = 0$
- $\mathtt{btocc}(\forall \tilde{i}.\mathsf{ch}^K(\tilde{T_c})) = \mathtt{btocc}(A) = 0$

And then, we also ask that for a canonical intermediate channel type, all the indexes for base types corresponding to the "base type index occurences" are an actual index variable (thus $n = 0$), and they are all distinct, and in the "left to right" order (thus, we use all the different index variable name exactly once in base types, and in a specific order). Moreover, as we are interested in a implementable procedure, a special focus is given to name of binding variables. We ask explicitely that in a canonical type, a binding name is never used twice in a type.

For example, this is a canonical channel type:

$$\forall i_1^1, i_2^1, i_3^1.\mathsf{ch}^{(i_1^1 + i_3^1)}(\mathsf{Nat}[0, i_1^1], \forall i_1^2.\mathsf{ch}^0(\mathsf{Nat}[0, i_1^2], \forall \cdot.\mathsf{ch}^3()), \mathsf{List}[0, i_2^1](\mathsf{List}[0, i_3^1](\alpha)), A)$$

The first quantification is over 3 index variables because there are exaclty 3 positions in this type in which we can put an index variable without crossing another quantifier. Then, those 3 variables are indeed ordered from left to right. All the subtypes are also canonical, and remark that base type variable and type variable are not taken in account in the counting of index variables.

Obviously, a canonical intermediate type is an intermediate type. For this intermediate type system, we will consider that all channel names have a canonical type. Moreover, in a context Γ, we will ask all types to be canonical.

Subtyping is defined as a restriction of the previous subtyping rule to the intermediate types, and the type system is given by Figure 16 and Figure 17. In this typing, and for the following of this section,

$$\frac{v : T_c \in \Gamma}{\phi; \Gamma \vdash v : T_c} \qquad \frac{}{\phi; \Gamma \vdash \mathtt{0} : \mathsf{Nat}[0,0]} \qquad \frac{\phi; \Gamma \vdash e : \mathsf{Nat}[0, I]}{\phi; \Gamma \vdash \mathtt{s}(e) : \mathsf{Nat}[0, I+1]}$$

$$\frac{}{\phi; \Gamma \vdash [] : \mathsf{List}[0,0](\mathcal{B})} \qquad \frac{\phi; \Gamma \vdash e : \mathcal{B}^1 \qquad \phi; \Gamma \vdash e' : \mathsf{List}[0, I](\mathcal{B}^2) \qquad \phi; \cdot \vdash \mathcal{B}^1, \mathcal{B}^2 \sqsubseteq \mathcal{B}}{\phi; \Gamma \vdash e :: e' : \mathsf{List}[0, I+1](\mathcal{B})}$$

$$\frac{}{\phi; \Phi; \Gamma \vdash \mathtt{tt} : \mathsf{Bool}} \qquad \frac{}{\phi; \Phi; \Gamma \vdash \mathtt{ff} : \mathsf{Bool}}$$

Figure 16: Intermediate Typing Rules for Expressions

$$\frac{}{\phi; \Gamma \vdash 0 \lhd 0} \qquad \frac{\phi; \Gamma \vdash P \lhd K_1 \qquad \phi; \Gamma \vdash Q \lhd K_2 \qquad \phi; \cdot \vDash K_1 + K_2 \leq K}{\phi; \Gamma \vdash P \mid Q \lhd K} \qquad \frac{\phi; \Gamma, a : T_c \vdash P \lhd K}{\phi; \Gamma \vdash (\nu a) P \lhd K}$$

$$\frac{\phi; \Gamma \vdash a : \forall \tilde{i}.\mathsf{ch}^K(\tilde{T_c}) \qquad (\phi, \tilde{i}); \Gamma, \tilde{v} : \tilde{T_c} \vdash P \lhd K' \qquad (\phi, \tilde{i}); \cdot \vdash K' \leq K}{\phi; \Gamma \vdash !a(\tilde{v}).P \lhd 0}$$

$$\frac{\phi; \Gamma \vdash a : \forall \tilde{i}.\mathsf{ch}^K(\tilde{T_c}) \qquad (\phi, \tilde{i}); \Gamma, \tilde{v} : \tilde{T_c} \vdash P \lhd K' \qquad (\phi, \tilde{i}); \cdot \vdash K' \leq K}{\phi; \Gamma \vdash a(\tilde{v}).P \lhd 0}$$

$$\frac{\phi; \Gamma \vdash a : \forall \tilde{i}.\mathsf{ch}^K(\tilde{T_c}) \qquad \phi; \Gamma \vdash \tilde{e} : \tilde{T_c}\{\tilde{J}/\tilde{i}\} \qquad \phi; \cdot \vDash K\{\tilde{J}/\tilde{i}\} \leq K'}{\phi; \Gamma \vdash \overline{a}\langle \tilde{e} \rangle \lhd K'} \qquad \frac{\phi; \Gamma \vdash P \lhd K}{\phi; \Gamma \vdash \mathtt{tick}.P \lhd K+1}$$

$$\frac{\phi; \Gamma \vdash v : \mathsf{Nat}[0,i] \qquad \phi; \Gamma \vdash P \lhd K_1 \quad \phi; \cdot \vdash K_1 \leq K \qquad \phi; \Gamma\{i+1/i\}, x : \mathsf{Nat}[0,i] \vdash Q \lhd K_2 \quad \phi; \cdot \vdash K_2 \leq K\{i+1/i\}}{\phi; \Gamma \vdash \mathtt{match}(v)\ \{\mathtt{0} \mapsto P; ;\ \mathtt{s}(x) \mapsto Q\} \lhd K}$$

$$\frac{\phi; \Gamma \vdash v : \mathsf{Nat}[0, i+n+1] \qquad \phi; \Gamma \vdash P \lhd K_1 \qquad \phi; \cdot \vdash K_1 \leq K \qquad \phi; \Gamma, x : \mathsf{Nat}[0, i+n] \vdash Q \lhd K_2 \qquad \phi; \cdot \vdash K_2 \leq K}{\phi; \Gamma \vdash \mathtt{match}(v)\ \{\mathtt{0} \mapsto P; ;\ \mathtt{s}(x) \mapsto Q\} \lhd K}$$

$$\frac{\phi; \Gamma \vdash v : \mathsf{List}[0,i](\mathcal{B}) \quad \phi; \Gamma \vdash P \lhd K_1 \quad \phi; \cdot \vdash K_1 \leq K \quad \phi; \Gamma\{i+1/i\}, x : \mathcal{B}, y : \mathsf{List}[0,i](\mathcal{B}) \vdash Q \lhd K_2 \quad \phi; \cdot \vdash K_2 \leq K\{i+1/i\}}{\phi; \Gamma \vdash \mathtt{match}(v)\ \{[] \mapsto P; ;\ x :: y \mapsto Q\} \lhd K}$$

$$\frac{\phi; \Gamma \vdash v : \mathsf{List}[0, i+n+1](\mathcal{B}) \quad \phi; \Gamma \vdash P \lhd K_1 \quad \phi; \cdot \vdash K_1 \leq K \quad \phi; \Gamma, x : \mathcal{B}, y : \mathsf{List}[0, i+n](\mathcal{B}) \vdash Q \lhd K_2 \quad \phi; \cdot \vdash K_2 \leq K}{\phi; \Gamma \vdash \mathtt{match}(v)\ \{[] \mapsto P; ;\ x :: y \mapsto Q\} \lhd K}$$

$$\frac{\phi; \Gamma \vdash e : \mathsf{Bool} \qquad \phi; \Gamma \vdash P \lhd K_1 \qquad \phi; \cdot \vdash K_1 \leq K \qquad \phi; \Gamma \vdash Q \lhd K_2 \qquad \phi; \cdot \vdash K_2 \leq K}{\phi; \Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \lhd K}$$

Figure 17: Intermediate Typing Rules for Processes

we consider only processes with well written pattern matching, meaning that pattern matching can only be done on base type variable (otherwise, the pattern matching is not useful...). (Remark that this intermediate type system is not used for its theoretical value, that is why we can ask those kind of restriction, since we are not interested in subject reduction for this intermediate type system). Note that thanks to this restriction, we can consider that the base type element in a pattern matching is a canonical type, and this helps us get rid of the usual rule for pattern matching when we need to add constraints in the branches.

Now, let us show that if a process is typable with those intermediate types, then it is typable for types of Figure 15.

Theorem 5. *If a process P is such that $\phi; \Gamma \vdash P \lhd K$ by the rules of Figure 16 and Figure 17, then, for any Γ_{sub} that is Γ where all type variable have been substituted by actual types, we have $\phi; \cdot; \Gamma_{sub} \vdash P \lhd K$ for the rules of Figure 7 and Figure 15.*

Proof. The proof is done by induction on $\phi; \Gamma \vdash P \lhd K$. The type variable does not cause any problem, since in a typing, if we can use a type variable then this type is never really inspected. When typing an expression, this is rather direct, using subtyping for expressions. Again, for typing rules for processes, a lot of cases are direct just by subtyping. Thus, we only detail the interesting cases .

- If the typing is

$$\dfrac{\dfrac{a : \forall \tilde{i}.\mathsf{ch}^K(\tilde{T}_c) \in \Gamma}{\phi; \Gamma \vdash a : \forall \tilde{i}.\mathsf{ch}^K(\tilde{T}_c)} \qquad (\phi, \tilde{i}); \Gamma, \tilde{v} : \tilde{T}_c \vdash P \lhd K' \qquad (\phi, \tilde{i}); \cdot \vdash K' \leq K}{\phi; \Gamma \vdash !a(\tilde{v}).P \lhd 0}$$

Note that with the rules of Figure 16, the only way to type a channel is by the axiom rule. Then, we can give the following type :

$$\dfrac{\dfrac{a : \forall \tilde{i}.\mathsf{ch}^K(\tilde{T}_{csub}) \in \Gamma_{sub} \qquad \phi; \cdot \vdash \forall \tilde{i}.\mathsf{ch}^K(\tilde{T}_{csub}) \sqsubseteq \forall \tilde{i}.\mathsf{in}^K(\tilde{T}_{csub})}{\phi; \cdot; \Gamma_{sub} \vdash a : \forall \tilde{i}.\mathsf{in}^K(\tilde{T}_{csub})} \qquad \dfrac{(\phi, \tilde{i}); \cdot; \Gamma_{sub}, \tilde{v} : \tilde{T}_{csub} \vdash P \lhd K' \qquad (\phi, \tilde{i}); \cdot \vdash K' \sqsubseteq K}{(\phi, \tilde{i}); \cdot; \Gamma_{sub}, \tilde{v} : \tilde{T}_{csub} \vdash P \lhd K}}{\phi; \cdot; \Gamma \vdash !a(\tilde{v}).P \lhd 0}$$

- If the typing is

$$\dfrac{\phi; \Gamma \vdash e : \mathsf{Nat}[0, i] \qquad \phi; \Gamma \vdash P \lhd K_1 \qquad \phi; \cdot \vdash K_1 \leq K \qquad \phi; \Gamma\{i+1/i\}, x : \mathsf{Nat}[0, i] \vdash Q \lhd K_2 \qquad \phi; \cdot \vdash K_2 \leq K\{i+1/i\}}{\phi; \Gamma \vdash \mathtt{match}(e)\ \{\mathtt{0} \mapsto P; ;\ \mathtt{s}(x) \mapsto Q\} \lhd K}$$

Then, we would like to give the following type :

$$\dfrac{\phi; \cdot; \Gamma_{sub} \vdash e : \mathsf{Nat}[0, i] \qquad \phi; (0 \leq 0); \Gamma_{sub} \vdash P \lhd K \qquad \phi; (i \geq 1); \Gamma_{sub}, x : \mathsf{Nat}[0-1, i-1] \vdash Q \lhd K}{\phi; \cdot; \Gamma_{sub} \vdash \mathtt{match}(e)\ \{\mathtt{0} \mapsto P; ;\ \mathtt{s}(x) \mapsto Q\} \lhd K}$$

From the typing $\phi; \Gamma \vdash P \lhd K_1$, obtaining $\phi; (0 \leq 0); \Gamma_{sub} \vdash P \lhd K$ is direct by induction hypothesis and weakening (cf Lemma 10, that could be proved for this type system as well). Now, by induction hypothesis we also obtain

$$\phi; \cdot; \Gamma_{sub}\{i+1/i\}, x : \mathsf{Nat}[0, i] \vdash Q \lhd K_2 \qquad \phi; \cdot \vdash K_2 \leq K\{i+1/i\}$$

Then, by index substitution (Lemma 12, that again could be proved for this type system as well), we have

$$\phi; \cdot; \Gamma_{sub}\{i{+}1/i\}\{i{-}1/i\}, x : \mathsf{Nat}[0, i{-}1] \vdash Q \lhd K_2\{i{-}1/i\} \qquad \phi; \cdot \vdash K_2\{i{-}1/i\} \leq K\{i{+}1/i\}\{i{-}1/i\}$$

Thus, by weakening we obtain:

$$\phi; (i \geq 1); \Gamma_{sub}\{i{+}1/i\}\{i{-}1/i\}, x : \mathsf{Nat}[0, i{-}1] \vdash Q \lhd K_2\{i{-}1/i\} \qquad \phi; (i \geq 1) \vdash K_2\{i{-}1/i\} \leq K\{i{+}1/i\}\{i{-}1/i\}$$

Then, as we have $i \geq 1$, we have $(i-1)+1 = i$. Moreover, $0-1 = 0$ by definition. So, by subtyping we obtain

$$\phi; (i \geq 1); \Gamma_{sub}, x : \mathsf{Nat}[0-1, i-1] \vdash Q \lhd K$$

Thus, we can conclude the proof, the typing above can be obtained. The other case of pattern matching with $n > 0$ is easier.

With this, we covered all the important cases. □

So, if we have a correct and complete procedure for this intermediate type system, we obtain indeed a bound on the complexity. Moreover, this new type system does not seem too restrictive. The main problem is that we cannot have too much information about the input. For example in order to have a more understandable type, we would like to suppose that an input list was of a size 2^i for some i. With this type system, this kind of assumption is not possible, thus we would need to use the logarithm in order to obtain a similar reasoning. In general, we can lose some information about the input of a function that may have been useful. Subtyping also become very restrained, we have no more input/output types and subtyping between canonical forms is not really useful, this is just equality everywhere. Note that in practice, because of canonical types, checking type equality is easy, since canonical forms simplify a lot the problems of α-renaming and reordering of quantifiers.

7 Parallel Complexity and Causal Complexity

In this section, we show that the parallel complexity defined in this report is equivalent to a causal complexity. Let us first present what is causal complexity in the π-calculus.

7.1 Causal Complexity

We present here a notion of causal complexity inspired by other works [8, 7, 6]. We explained before with the canonical form that a process can be described by a set of names and a multiset of guarded processes, when working up to congruence. For causal complexity, we consider more structure for processes. The idea is to see a process as a set of names and a binary tree where leaves are guarded processes and a node means parallel composition. So formally, instead of using the previous congruence relation, we use the *tree congruence*. This is defined as the least congruence relation $\equiv_t$ closed under:

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

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

So, this tree congruence can indeed move names as before, but it preserves the tree-shape of a process. With this intuition, we can redefine the semantics in order to preserve this tree structure. The rules are described in Figure 18. So, for example, from a process P seen as a tree with at position p a replicated input $!a(\tilde{v}).Q$ and at position p' an output $\overline{a}\langle\tilde{e}\rangle$, we obtain the process P', the same tree as P, with at position p the same replicated input but at position p' we have the tree corresponding to the process $Q[\tilde{v} := \tilde{e}]$. This reduction step is annotated by a *location* $\tau\langle p, p'\rangle$ in order to remember at which positions the modification happened, and what was the action performed (here τ means a communication). In the same way, we can define a reduction annotated by a `tick`, or a match. An alternative presentation of this semantics, closer to the one in [8], with a labelled transition system, is also possible. As for the standard reduction in π-calculus, both semantics (the one with the congruence and the one with the labelled transition system) are equivalent. So, we chose to present this one because it is easier to use in proofs.

The goal of this semantics is first to preserve the tree structure in a reduction step, and second to remember when doing a reduction step where the modification occurs exactly in the tree. Then, we can define a causality relation between locations. The idea is that a location ℓ_1 *causes* a location ℓ_2 when the reduction step with location ℓ_2 could not have happened without the reduction step with location ℓ_1.

Formally, we define a location by the following grammar.

$$p := \epsilon \mid 0 \cdot p \mid 1 \cdot p \qquad \ell := p;\texttt{tick} \mid p;\mathsf{if} \mid \tau\langle p, p'\rangle$$

The intuition is that a reduction $P \to P'$ with location $p;\texttt{tick}$ removed the top `tick` of the guarded process of P at position p. A reduction $P \to P'$ with location $\tau\langle p, p'\rangle$ is a communication between the input process at position p and the output process at position p'. Finally, the action $p;\mathsf{if}$ is for the position of a pattern-matching reduction. Then, we can define a causality relation $\ell \prec_c \ell'$ between locations:

Definition 14. *The causality relation $\ell \prec_c \ell'$ between locations is defined by:*

- $p;\texttt{tick} \prec_c p';\texttt{tick}$ *when p is a prefix of p'*
- $p;\texttt{tick} \prec_c \tau\langle p_0, p_1\rangle$ *when p is a prefix of p_0 or p_1*
- $\tau\langle p_0, p_1\rangle \prec_c p;\texttt{tick}$ *when p_1 is a prefix of p*
- $\tau\langle p_0, p_1\rangle \prec_c \tau\langle p'_0, p'_1\rangle$ *when p_1 is a prefix of p'_0 or p'_1.*

By extension, we will sometime say that a location $\ell \prec_c p$ when $\ell \prec_c p;\texttt{tick}$.

And for if locations, they behave as a `tick` location. The important point is that a τ location causes another location ℓ when the output position is a prefix of the positions in ℓ. Indeed, for a communication with a non-replicated input, the input position becomes a 0 thus it cannot cause anything, and for a

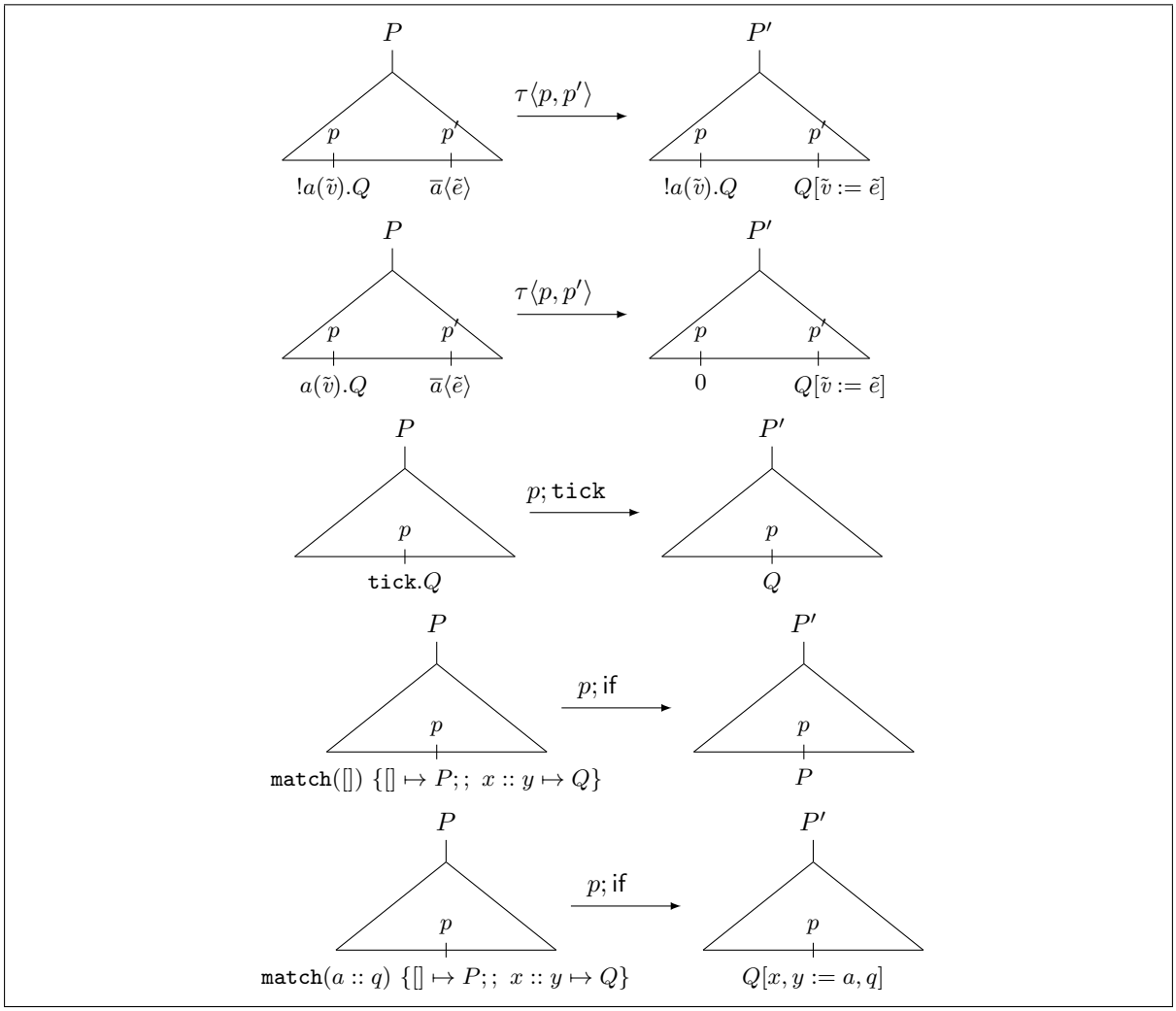

Figure 18: Semantics for Causal Complexity

communication with a replicated input, we consider that two calls to the same replicated input are independent of each other. Then, the important point is that two reductions with independent locations could be done in any order and it would not change the final tree.

With this definition of causality, intuitively we can define causal complexity as the maximal number of tick in all the chains of causality in a computation.

Definition 15 (Computation)**.** *A* computation *from a process P is a sequence $(P_i, \ell_i)_{i \leq N}$ such that $P_0 = P$ and $P_i \xrightarrow{\ell_i} P_{i+1}$ for $i < N$ and P_N is in normal form for reductions.*

Definition 16 (Causal Complexity)**.** *In a computation $(P_i, \ell_i)_{i \leq N}$, we say that ℓ_i* depends *on ℓ_j, noted $\ell_i \prec \ell_j$ when $i < j$ and $\ell_i \prec_c \ell_j$. Then, the causal complexity of this computation is given by the maximal number of tick locations in all the chains of $\prec^\star$, the reflexive and transitive closure of $\prec$. Then, the causal complexity of a process P is defined as the maximal causal complexity over all computations from P.*

We can now prove equivalence between this two notions.

7.2 Parallel Complexity $\geq$ Causal Complexity

In this section, we consider causal complexity, and we show that the parallel complexity is always greater than causal complexity. Formally, we prove the following lemma.

Lemma 17. *Let P be a process. Let $(P_i, \ell_i)_{i \leq N}$ be a computation from P with causal complexity K. Then, the global parallel complexity of P is greater than K.*

In order to do that, we show that we can do the same computation with our semantics with annotated processes. Let us take a process P seen as a tree. By definition, each leaf of P is a guarded process G. For each leaf, we replace the guarded process G by $0 : G$, and we call P' this annotated process. By the definition of congruence for annotated processes, we have $P \equiv P'$. Then, we will work with this tree representation for annotated processes.

Definition 17 (Tree Representation of Annotated Processes)**.** *We consider annotated trees such that a set of names are given at the beginning, nodes represent parallel composition and leaves are processes of the shape $n : G$ with G guarded. Such a tree represents indeed an annotated process.*

Then, we say that an annotated process P' seen as a tree is an annotation *of a process P seen as a tree if P' and P have exactly the same shape, and each leaf G of P is a leaf $n : G$ for P'.*

So, by definition P' is an annotation of P. We can then prove the following lemma:

Lemma 18 (Causal Reduction and Annotation)**.** *Suppose that $P \xrightarrow{\ell} Q$. Then, for any P' annotation of P, we have $P' \Rightarrow_p Q'$ where Q' is an annotation of Q.*

Moreover, we have:

- *If $\ell = p; \texttt{tick}$ then Q' has the same annotation as P' for all leaves at a position p' such that p is not a prefix of p'. For all the other leaves, Q' has the annotation $n + 1$ where n is the annotation for the leaf at position p in P'*
- *If $\ell = p; \mathsf{if}$ then Q' has the same annotation as P' for all leaves at a position p' such that p is not a prefix of p'. For all the other leaves, Q' has the annotation n where n is the annotation for the leaf at position p in P'*
- *If $\ell = \tau\langle p, p' \rangle$ then Q' has the same annotation as P' for all leaves at position q such that p' is not a prefix of q. For all the other leaves, Q' has the annotation $max(n, m)$ where n is the annotation for the leaf at position p in P' and m is the annotation for the leaf at position p' in P'.*

Proof. The proof is done by case analysis on the rules of Figure 18. All cases are rather direct. The idea is to always keep the same tree shape in the reduction $\Rightarrow_p$, and then to make the names go up and the annotations go down after doing this reduction using the congruence rules. Formally, the names can go up with $((\nu a)P) \mid Q \equiv (\nu a)(P \mid Q)$ (always possible by α-renaming) and $n : ((\nu a)P) \equiv (\nu a)(n : P)$, and then the annotations can go down with $n : (P \mid Q) \equiv n : P \mid n : Q$. Then, with the shape of the reduction $\Rightarrow_p$ we can indeed see that the annotations correspond indeed to the one given in this lemma. □

With this lemma, we can start from P' the annotation of P, and simulate the computation. Now we only need to show that the annotations correspond to the number of ticks in a chain of dependency. Formally, we prove the following lemma.

Lemma 19. *Let $(P'_i, \ell_i)_{i \leq N}$ be the computation given above. Then, for all $i \leq N$, the annotation of a guarded process G at position p in P'_i is an upper bound of the maximal number of* `tick` *locations in all chains of $\prec^\star$ for the locations $\ell_0, \ldots, \ell_{i-1}$ such that the last location ℓ of this chain is such that $\ell \prec_c p$*

We prove this by induction on i.

- This is true for $i = 0$ since P' is P annotated with zeros everywhere.
- Let $i < N$. Suppose that this is true for P'_i, the annotation of P_i. Let us look at the reduction $P_i \xrightarrow{\ell_i} P_{i+1}$.
  - If $\ell_i = p; \mathsf{if}$. Then by induction hypothesis, the annotation n for the pattern matching bound the maximal number of `tick` locations in all chains of $\prec^\star$ for the locations $\ell_0, \ldots, \ell_{i-1}$ such that the last location ℓ is such that $\ell \prec_c p$. Let us look at P'_{i+1} given by lemma 18. For all the positions in P'_{i+1} with p not a prefix, dependency did not change since $\ell_i = p; \mathsf{if}$ do not cause those positions. As annotations did not change either, the hypothesis is still correct. For the new positions in the tree with p as a prefix, all the annotations are n. All chain of causality $\prec^\star$ with the locations $\ell_0, \ldots, \ell_i$ are either chains that do not contain ℓ_i and that caused p and so n is a bound by induction hypothesis, either it contains ℓ_i and so it was a chain in causal relation with p and so n is a bound.
  - If $\ell_i = p; \texttt{tick}$. Then by induction hypothesis, the annotation n for the tick corresponds to the maximal number of `tick` locations in all chains of $\prec^\star$ for the locations $\ell_0, \ldots, \ell_{i-1}$ that are also in causality $\prec_c$ with p. Let us look at P'_{i+1} given by lemma 18. For all the positions in P'_{i+1} with p not a prefix, dependency did not change since $\ell_i = p; \mathsf{if}$ do not cause those positions. As annotations did not change either, the hypothesis is still correct. For the new positions in the tree with p as a prefix, all the annotations are n. All chain of causality $\prec^\star$ with the locations $\ell_0, \ldots, \ell_i$ are either chains that do not contain ℓ_i and that caused p and so $n + 1$ is a bound because n is a bound by induction hypothesis, either it contains ℓ_i and so it was a chain in causal relation with p and so $n + 1$ is a bound as n is a bound on the causality between ℓ_0 and ℓ_{i-1} and the last location ℓ_i add one to the complexity.
  - If $\ell_i = \tau\langle p, p' \rangle$. By induction hypothesis, the annotation n for the input and m for the output are bounds on some chains of causality on $\ell_0, \ldots, \ell_{i-1}$. Let us look at P'_{i+1} given by Lemma 18. For all positions that are not a prefix of p', nothing changed so the hypothesis is still true. Let us look at positions with p' as a prefix. All the annotations for those positions are $max(n, m)$. Let us look at chains of causality on $\ell_0, \ldots, \ell_i$ that ends with a location that causes those positions. The new chains that were not in the previous hypothesis are the one that finish with ℓ_i. For those chains, either they cause ℓ_i because they cause p or because they cause p'. In both way, n or m were a bound on the number of ticks by induction hypothesis. So, $max(n, m)$ is indeed a bound on the number of ticks for all those chains.

  This concludes this case.

This concludes the proof. So, in the end, the annotation in position p in P_N is a bound on chains of causality that also cause p. Moreover, for any chains of causality, this chain causes its last position (or output position by definition of causality). So, all chains of causality are bounded by at least one of the annotations, so the maximum over all annotations is a bound on the causal complexity. This gives us directly that global parallel complexity is greater than causal complexity.

7.3 Causal Complexity $\geq$ Parallel Complexity

Let us work on the reverse. In order to do that, we will restrict a bit the congruence $\equiv$ for annotated processes and expand the semantics $\Rightarrow_p$ in order to work with trees. So, as before, we can define a tree congruence $\equiv_t$ for annotated processes, with the base rules

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

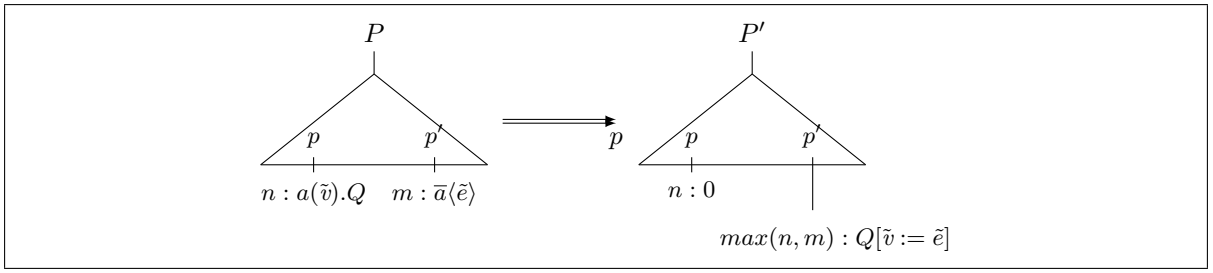

Figure 19: Semantics for Causal Complexity

$(\nu a)(P \mid Q) \equiv_t (\nu a)P \mid Q$ (when a is not free in Q) $\quad (\nu a)(P \mid Q) \equiv_t P \mid (\nu a)Q$ (when a is not free in P)

$n : (P \mid Q) \equiv n : P \mid n : Q \quad n : (m : P) \equiv (n + m) : P \quad (\nu a)(n : P) \equiv n : ((\nu a)P) \quad 0 : P \equiv P$

And then we define the semantics $\Rightarrow_p$ exactly as before but with trees instead of simple processes in parallel. An example is given in Figure 19

As before, any annotated process can be written in a tree representation as in Definition 17 using the tree congruence rule $\equiv_t$ for annotated processes. So, from this, it is rather direct than this semantics defined with tree $\Rightarrow_p$ is equivalent to the previously defined $\Rightarrow_p$, in the sense that they give the same complexity. (It relies in particular on the fact that congruence does not change the parallel complexity). And now, we can work on this parallel complexity with tree representation. Formally, we want to prove:

Lemma 20. *If P is a process without annotation, and if $P(\Rightarrow_p)^*Q$, with Q in normal form for $\Rightarrow_p$ and $\mathcal{C}_\ell(Q) = K$, then there is a computation $(P_i, \ell_i)_{i \leq N}$ from P with causal complexity greater than K.*

So, by definition of causal complexity and parallel complexity, this will indeed show that the causal complexity is greater than the parallel complexity.

As we only need to prove this lemma for process without annotation, we can take an additional hypothesis on annotated processes: we consider in the following that annotations appear exclusively at the "top-level" of a process, so there are no annotation in the subprocess P in $a(\tilde{v}).P$, in $\texttt{tick}.P$, in $!a(\tilde{v}).P$ or in the subprocesses of a pattern matching. We can take this hypothesis because if a process P satisfies this hypothesis and $P \Rightarrow_p Q$ then Q also satisfies this hypothesis. And of course, processes without annotation satisfy this hypothesis. Note that when we say that, we do not consider $0 : P$ as a real annotation since it can be removed by congruence. We start by the following definition.

Definition 18 (Removing Annotation)**.** *Let P be a tree representation of an annotated process. We define $\mathsf{forget}(P)$ as the tree P where leaves $n : G$ are replaced by G.*

Note that this definition makes sense only because of the previous hypothesis, otherwise we would need to apply the forgetful function to G.

We can then easily show the following lemma.

Lemma 21 (Forget Reductions)**.** *Suppose that $P \Rightarrow_p Q$. Then, we have $\mathsf{forget}(P) \xrightarrow{\ell} \mathsf{forget}(Q)$ for some location ℓ.*

Proof. The proof is direct because we modified $\Rightarrow_p$ in order to obtain immediately this. Note that the reduction $\texttt{tick}.P \Rightarrow_p 1 : P$, or more generally $n : \texttt{tick}.P \Rightarrow_p (n + 1) : P$ here corresponds indeed to removing a tick with the forgetful function. □

With this lemma, we can start from $\mathsf{forget}(P)$ and simulate the reduction $(\Rightarrow_p)^*$. Now we only need to show that the annotations are bounded by the number of ticks in a chain of dependency. We show the following lemma:

Lemma 22. *If P is a process without annotation, and if $P(\Rightarrow_p)^*Q$, then there is a computation $(P_i, \ell_i)_{i \leq N}$ from $P = \mathsf{forget}(P)$ to $\mathsf{forget}(Q)$. Moreover, for each leaf $n : G$ at position p in Q, there is a chain of causality $\prec^\star$ with at least n ticks that ends with a location ℓ such that $\ell \prec_c p$.*

Note that from this lemma we can deduce immediately Lemma 20. We prove this by induction on $P(\Rightarrow_p)^*Q$

- If this relation is the reflexive one, and $P = Q$, this is direct because all annotations are equal to 0.
- Now suppose we have $P(\Rightarrow_p)^* R \Rightarrow_p Q$. By induction hypothesis, there is a computation $(P_i, \ell_i)_{i \leq N}$ from $\mathsf{forget}(P)$ to $\mathsf{forget}(R)$, with the expected chains of causality. We now proceed by case analysis on $R \Rightarrow_p Q$.
  - If this reduction is a pattern matching reduction at position p. Then, we have $\mathsf{forget}(R) \xrightarrow{p;\mathsf{if}} \mathsf{forget}(Q)$. Let us take a leaf $n : G$ at position p' of Q. If p is not a prefix of p', then this leaf was also in R. So, by induction hypothesis, we obtain the desired chain of causality. If p is a prefix of p', then this n was also the annotation for the position p in R. So, by induction hypothesis for R, there is a chain of causality $\prec^\star$ with at least n ticks that ends with a location ℓ such that $\ell \prec_c p$. By definition, it means that this last location $\ell \prec_c p; \mathsf{if}$. So, this gives us a chain of causality with at least n ticks that ends with a location that cause p' in Q. This concludes this case.
  - If this reduction is a tick reduction at position p. Then, we have $\mathsf{forget}(R) \xrightarrow{p;\mathtt{tick}} \mathsf{forget}(Q)$. Let us take a leaf $n : G$ at position p' of Q. If p is not a prefix of p', then this leaf was also in R. So, by induction hypothesis, we obtain the desired chain of causality. If p is a prefix of p', then $n - 1$ was the annotation for the position p in R. So, by induction hypothesis for R, there is a chain of causality $\prec^\star$ with at least n ticks that ends with a location ℓ such that $\ell \prec_c p$. By definition, it means that this last location $\ell \prec_c p; \mathtt{tick}$. So, this gives us a chain of causality with at least $n+1$ ticks that ends with a location that causes p' in Q. This concludes this case.
  - If this reduction is a synchronization with input at position p and output at position p'. Then, we have $\mathsf{forget}(R) \xrightarrow{\tau\langle p,p'\rangle} \mathsf{forget}(Q)$. Let us take a leaf $n : G$ at position q of Q. If p' is not a prefix of q, then this leaf was also in R (except for the position p in the case of non-replicated input where the guarded process changes but not the annotation, still the following reasoning works). So, by induction hypothesis, we obtain the desired chain of causality. If p' is a prefix of q, then, we have $n = max(n_0, n_1)$ with n_0 the annotation for the input position p in R and n_1 the annotation for the output position p' in R. Let us say, by symmetry, that n_0 is the maximum between those two. So, by induction hypothesis for R, there is a chain of causality $\prec^\star$ with at least n_0 ticks that ends with a location ℓ such that $\ell \prec_c p$. By definition, it means that this last location $\ell \prec_c \tau\langle p, p'\rangle$. So, this gives us a chain of causality with at least n_0 ticks that ends with a location that cause q in Q. This concludes this case.

  This concludes the proof.

So, we have indeed that causal complexity is greater than parallel complexity.
From this, we have the equivalence between causal complexity and parallel complexity.

References

[1] Selim G. Akl. *Encyclopedia of Parallel Computing*, chapter Bitonic Sort, pages 139–146. Springer US, Boston, MA, 2011.

[2] Martin Avanzini and Ugo Dal Lago. Automating sized-type inference for complexity analysis. *Proceedings of the ACM on Programming Languages*, 1(ICFP):43, 2017.

[3] Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on*, pages 133–142. IEEE, 2011.

[4] Ankush Das, Jan Hoffmann, and Frank Pfenning. Parallel complexity analysis with temporal session types. *Proc. ACM Program. Lang.*, 2(ICFP):91:1–91:30, 2018.

[5] Ankush Das, Jan Hoffmann, and Frank Pfenning. 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*, pages 305–314. ACM, 2018.

[6] Pierpaolo Degano, Fabio Gadducci, and Corrado Priami. Causality and replication in concurrent processes. In *Perspectives of System Informatics*, pages 307–318. Springer Berlin Heidelberg, 2003.

[7] Pierpaolo Degano and Corrado Priami. Causality for mobile processes. In *Automata, Languages and Programming*, pages 660–671. Springer Berlin Heidelberg, 1995.

[8] Romain Demangeon and Nobuko Yoshida. Causal computational complexity of distributed processes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 344–353. ACM, 2018.

[9] Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of pspace. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 121–131. ACM, 2008.

[10] E. Hainry, J.-Y. Marion, and R. Péchoux. Type-based complexity analysis for fork processes. In *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7794 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2013.

[11] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2012.

[12] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14:1–14:62, 2012.

[13] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Resource aware ML. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 781–786. Springer, 2012.

[14] Jan Hoffmann and Martin 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*, volume 6012 of *Lecture Notes in Computer Science*, pages 287–306. Springer, 2010.

[15] Jan Hoffmann and Zhong Shao. Automatic static cost analysis for parallel programs. In Jan Vitek, editor, *Programming Languages and Systems*, pages 132–157, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[16] Martin Hofmann. Linear types and non-size-increasing polynomial time computation. *Information and Computation*, 183(1):57–85, 2003.

[17] Martin Hofmann and Steffen 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*, pages 185–197. ACM, 2003.

[18] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, sep 1999.

[19] J.-Y. Marion. A type system for complexity flow analysis. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 123–132. IEEE Computer Society, 2011.

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