Light Logics and Higher-Order Processes

Ugo Dal Lago
Università di Bologna
INRIA Sophia Antipolis
dallago@cs.unibo.it

Simone Martini
Università di Bologna
INRIA Sophia Antipolis
martini@cs.unibo.it

Davide Sangiorgi
Università di Bologna
INRIA Sophia Antipolis
sangio@cs.unibo.it

We show that the techniques for resource control that have been developed in the so-called "light logics" can be fruitfully applied also to process algebras. In particular, we present a restriction of Higher-Order π-calculus inspired by Soft Linear Logic. We prove that any soft process terminates in polynomial time. We argue that the class of soft processes may be naturally enlarged so that interesting processes are expressible, still maintaining the polynomial bound on executions.

1 Introduction

A term terminates if all its reduction sequences are of finite length. As far as programming languages are concerned, termination means that computation in programs will eventually stop. In computer science, termination has been extensively investigated in sequential languages, where strong normalization is a synonym more commonly used.

Termination is however interesting also in concurrency. While large concurrent systems often are supposed to run forever (e.g., an operating system, or the Internet itself), single components are usually expected to terminate. For instance, if we query a server, we may want to know that the server does not go on forever trying to compute an answer. Similarly, when we load an applet we would like to know that the applet will not run forever on our machine, possibly absorbing all the computing resources. In general, if the lifetime of a process can be infinite, we may want to know that the process does not remain alive simply because of nonterminating internal activity, and that, therefore, the process will eventually accept interactions with the environment.

Another motivation for studying termination in concurrency is to exploit it within techniques aimed at guaranteeing properties such as responsiveness and lock-freedom [9], which intuitively indicate that certain communications or synchronizations will eventually succeed (possibly under some fairness assumption). In message-passing languages such as those in the π-calculus family (Join Calculus, Higher-Order π-calculus, Asynchronous π-calculus, etc.) most liveness properties can be reduced to instances of lock-freedom. Examples, in a client-server system, are the liveness properties that a client request will eventually be received by the server, or that a server, once accepted a request, will eventually send back an answer.

However, termination alone may not be satisfactory. If a query to a server produces a computation that terminates after a very long time, from the client point of view this may be the same as a nonterminating (or failed) computation. Similarly, an applet loaded on our machine that starts a very long computation, may engender an unacceptable consumption of local resources, and may possibly be considered a "denial of service" attack. In other words, without precise bounds on the time to complete a computation, termination may be indistinguishable from nontermination.

Type disciplines are among the most general techniques to ensure termination of programs. Both in the sequential and in the concurrent case, type systems have been designed to characterize classes of terminating programs. It is interesting that, from the fact that a program has a type, we may often extract information on the structure of the program itself (e.g., for the simple types, the program has no self applications). If termination (or, more generally, some property of the computation) is the main interest, it is only this structure that matters, and not the specifics of the types. In this paper we take this perspective, and apply to a certain class of programs (Higher-Order π-calculus terms) the structural restrictions suggested by the types of Soft Linear Logic [10], a fragment of Linear Logic [7] characterizing polynomial time computations.

Essential contribution of Linear Logic has been the *refinement* it allows on the analysis of computation. The (previously atomic) step of function application is decomposed into a duplication phase (during which the argument is duplicated the exact number of times it will be needed during the computation), followed by the application of a *linear* function (which will use each argument exactly once). The emphasis here is not on restricting the class of programs—in many cases, any traditional program (e.g., any λ-term, even a divergent one) could be annotated with suitable *scope information* (*boxes*, in the jargon) in such a way that the annotated program behaves as the original one. However, the new annotations embed information on the computational behavior that was unexpressed (and inexpressible) before. In particular, boxes delimit those parts of data that will be (or may be) duplicated or erased during computation.

It is at this stage that one may apply *restrictions*. By building on the scopes exposed in the new syntax, we may restrict the computational behavior of a term. In the sequential case several achievements have been obtained via the so-called *light logics* [8, 2, 10], which allow for type systems for λ-calculus exactly characterizing several complexity classes (notably, elementary time, polynomial type, polynomial space, logarithmic space). This is obtained by limitations on the way the scopes (boxes) may be manipulated. For the larger complexity classes (e.g., elementary time) one forbids that during computation one scope may enter inside another scope (their nesting depth remains constant). For smaller classes (e.g., polynomial time) one also forbids that a duplicating computation could drive another duplication. The exact way this is obtained depends on the particular discipline (either à la Light Linear Logic, or à la Soft Linear Logic).

The aim of this paper is to apply for the first time these technologies to the concurrent case, in particular to Higher-Order π-calculus [12]. We closely follow the pattern we have delineated above. First, we introduce (higher-order) processes, which we then annotate with explicit scopes, where the new construct "!" marks duplicable entities. This is indeed a refinement, and not a restriction — any process in the first calculus may be simulated by an annotated one. We then introduce our main object of study — annotated processes restricted with the techniques of Soft Linear Logic. We show that the number of internal actions performed by processes of this calculus is polynomially bounded (Section 4), a property that we call *feasible termination*. Moreover, an extension of the calculus capturing a natural example will be presented (Section 5).

We stress that we used in the paper a pragmatic approach — take from the logical side tools and techniques that may be suitable to obtain general bounds on the computing time of processes. We are not looking for a general relation between logical systems and process algebras that could realize a form of Curry-Howard correspondence among the two. That would be a much more ambitious goal, for which other techniques — and different success criteria — should be used.

Related Work A number of works have recently studied type systems that ensure termination in mobile processes, e.g. [14, 3, 4]. They are quite different from the present paper. First, the techniques employed are measure-based techniques, or logical relations, or combinations of these, rather than techniques inspired by linear logics, as done here. Secondly, the objective is pure termination, whereas here we aim at deriving polynomial bounds on the number of steps that lead to termination. (In some of the measure-based systems bounds can actually be derived, but they are usually exponential with respect to integer annotations that appear in the types.) Thirdly, with the exception of [4], all works analyse name-passing calculi such as the π-calculus, whereas here we consider higher-order calculi in which terms of the calculus are exchanged instead of names.

Linear Logic has been applied to mobile processes by Ehrhard and Laurent [5], who have studied encodings of π-calculus-like languages into Differential Interaction Nets [6], an extension of the Multiplicative Exponential fragment of Linear Logic. The encodings are meant to be tests for the expressiveness of Differential Interaction Nets; the issue of termination does not arise, as the process calculi encoded are finitary. Amadio and Dabrowski [1] have applied ideas from term rewriting to a π-calculus enriched with synchronous constructs à la Esterel. Computation in processes proceeds synchronously, divided into cycles called instants. A static analysis and a finite-control condition guarantee that, during each instant, the size of a program and the times it takes to complete the instant are polynomial on the size of the program and the input values at the beginning of the instant.

2 Higher-Order Processes

This section introduces the syntax and the operational semantics of processes. We call $\mathbf{HO}\pi$ the calculus of processes we are going to define (it is the calculus $\mathbf{HO}\pi^{\texttt{unit},\rightarrow,\diamond}$ in [12]). In $\mathbf{HO}\pi$ the values exchanged in interactions can be first-order values and higher-order values, i.e., terms containing processes. For economy, the only first-order value employed is the unit value $\star$, and the only higher-order values are parametrised processes, called abstractions (thus we forbid direct communication of processes; to communicate a process we must add a dummy parameter to it). The process constructs are nil, parallel composition, input, output, restriction, and application. Application is the destructor for abstraction: it allows us to instantiate the formal parameters of an abstraction. Here is the complete grammar:

$$\begin{aligned} P &::= \mathbf{0} \mid P \,||\, P \mid a(x).P \mid \overline{a}\langle V\rangle.P \mid (\nu a)P \mid VV; \\ V &::= \star \mid x \mid \lambda x.P; \end{aligned}$$

where a ranges over a denumerable set $\mathscr{C}$ of channels, and x over the denumerable set of variables. Input, restriction, and abstractions are binding constructs, and give rise in the expected way to the notions of free and bound channels and of free and bound variables, as well as of α-conversion.

Ill-formed terms such as $\star\star$ can be avoided by means of a type systems. The details are standard and are omitted here; see [12].

The operational semantics, in the reduction style, is presented in Figure 1, and uses the auxiliary relation of *structural congruence*, written $\equiv$. This is the smallest congruence closed under the following rules:

$$\begin{aligned} P &\equiv Q \text{ if } P \text{ and } Q \text{ are } \alpha\text{-equivalent;} \\ P \,||\, (Q \,||\, R) &\equiv (P \,||\, Q) \,||\, R; \\ P \,||\, Q &\equiv Q \,||\, P; \\ (\nu a)((\nu b)P) &\equiv (\nu b)((\nu a)P); \\ ((\nu a)P \,||\, Q) &\equiv ((\nu a)P) \,||\, Q \text{ if } a \text{ is not free in } Q; \end{aligned}$$

Unlike other presentations of structural congruence, we disallow the garbage-collection laws $P \,||\, \mathbf{0} \equiv P$ and $(\nu a)\mathbf{0} \equiv a$, which are troublesome for our resource-sensitive analysis. The reduction relation is written $\rightarrow_{\mathsf{P}}$, and is defined on processes without free variables.

$$\frac{}{\overline{a}\langle V\rangle.P \mid\mid a(x).Q \rightarrow_{\mathsf{P}} P \mid\mid Q[x/V]} \qquad \frac{}{(\lambda x.P)V \rightarrow_{\mathsf{P}} P[x/V]}$$

$$\frac{P \rightarrow_{\mathsf{P}} Q}{P \mid\mid R \rightarrow_{\mathsf{P}} Q \mid\mid R} \qquad \frac{P \rightarrow_{\mathsf{P}} Q}{(\nu a)P \rightarrow_{\mathsf{P}} (\nu a)Q} \qquad \frac{P \equiv Q \quad Q \rightarrow_{\mathsf{P}} R \quad R \equiv S}{P \rightarrow_{\mathsf{P}} S}$$

Figure 1: The operational semantics of **HO**π processes.

In general, the relation $\rightarrow_{\mathsf{P}}$ is nonterminating. The prototypical example of a nonterminating process is the following process *OMEGA*:

$$OMEGA = (\nu a)(DELTA\star \mid\mid \overline{a}\langle DELTA\rangle), \qquad \text{where} \qquad DELTA = \lambda y.(a(x).(x\star \mid\mid \overline{a}\langle x\rangle)).$$

Indeed, it holds that $OMEGA \rightarrow^2_{\mathsf{P}} OMEGA$. Variants of the construction employed for *OMEGA* can be used to show that process recursion can be modelled in **HO**π. An example of this construction is the following *SERVER* process. It accepts a request y on channel b and forwards it along c. After that, it can handle another request from b. In contrast to *OMEGA*, *SERVER* is terminating, because there is no infinite reduction sequence starting from *SERVER*. Yet hand, the number of requests *SERVER* can handle is unlimited, i.e., *SERVER* can be engaged in an infinite sequence of interactions with its environment.

$$\begin{aligned} SERVER &= (\nu a)(COMP\star \mid\mid \overline{a}\langle COMP\rangle); \\ COMP &= \lambda z.(a(x).(b(y).\overline{c}\langle y\rangle.\overline{a}\langle x\rangle \mid\mid x\star)). \end{aligned}$$

A remark on notation: in this paper, ! is the Linear Logic operator (more precisely, an operator derived from Linear Logic), and should not be confused with the replication operator often used in process calculi such as the π-calculus.

3 Linearizing Processes

Linear Logic can be seen as a way to decompose the type of functions $A \rightarrow B$ into a refined type $!A \multimap B$. Since the argument (in A) may be used several (or zero) times to compute the result in B, we first turn the input into a duplicable (and erasable) object (of type $!A$). We now duplicate (or erase) it the number of times it is needed, and finally we use each of the copies exactly once to obtain the result (this is the linear function space $\multimap$). The richer language of types (with the new constructors ! and $\multimap$) is matched by new term constructs, whose goal is to explicitly enclose in marked scopes (boxes) those subterms that may be erased or duplicated. In the computational process we described above, there are three main ingredients: (i) the mark on a duplicable/erasable entity; (ii) its actual duplication/erasure; (iii) the linear use of the copies. For reasons that cannot be discussed here (see Wadler's [13] for the notation we will use) we may adopt a syntax where the second step (duplication) is not made fully explicit (thus resulting in a simpler language), and where the crucial distinction is made between linear functions (denoted by the usual syntax $\lambda x.P$ — but interpreted in a strictly linear way: x occurs once in P), and nonlinear functions, denoted with $\lambda !x.P$, where the x may occur several (or zero) times in P. When a nonlinear function is applied, its actual argument will be duplicated or erased. We enclose the argument in a box to record this fact, using an eponymous unary operator ! also on terms. Since we want to control the computational behavior of duplicable entities, a term in a !-box is protected and cannot be reduced. Only when it will

$$\frac{}{!\Gamma \vdash_{\mathsf{P}} \mathbf{0}} \qquad \frac{\Gamma, !\Lambda \vdash_{\mathsf{P}} P \quad \Delta, !\Lambda \vdash_{\mathsf{P}} Q}{\Gamma, \Delta, !\Lambda \vdash_{\mathsf{P}} P \,||\, Q} \qquad \frac{\Gamma, x \vdash_{\mathsf{P}} P}{\Gamma \vdash_{\mathsf{P}} a(x).P}$$

$$\frac{\Gamma, !x \vdash_{\mathsf{P}} P}{\Gamma \vdash_{\mathsf{P}} a(!x).P} \qquad \frac{\Gamma, !\Lambda \vdash_{\mathsf{V}} V \quad \Delta, !\Lambda \vdash_{\mathsf{P}} P}{\Gamma, \Delta, !\Lambda \vdash_{\mathsf{P}} \overline{a}\langle V\rangle.P} \qquad \frac{\Gamma \vdash_{\mathsf{P}} P}{\Gamma \vdash_{\mathsf{P}} (\nu a)P}$$

$$\frac{\Gamma, !\Lambda \vdash_{\mathsf{V}} V \quad \Delta, !\Lambda \vdash_{\mathsf{V}} W}{\Gamma, \Delta, !\Lambda \vdash_{\mathsf{P}} VW} \qquad \frac{}{!\Gamma \vdash_{\mathsf{V}} \star} \qquad \frac{}{!\Gamma, x \vdash_{\mathsf{V}} x}$$

$$\frac{}{!\Gamma, !x \vdash_{\mathsf{V}} x} \qquad \frac{\Gamma, x \vdash_{\mathsf{P}} P}{\Gamma \vdash_{\mathsf{V}} \lambda x.P} \qquad \frac{\Gamma, !x \vdash_{\mathsf{P}} P}{\Gamma \vdash_{\mathsf{V}} \lambda !x.P} \qquad \frac{!\Gamma \vdash_{\mathsf{V}} V}{!\Gamma \vdash_{\mathsf{V}} !V}$$

Figure 2: Processes and values in **LHO**π.

be fed to a (nonlinear) function, and thus (transparently) duplicated, its box will be opened (the mark ! disappears) and the content will be reduced.

The constructs on *terms* arising from Linear Logic have a natural counterpart in higher-order processes, where communication and abstraction play a similar role. This section introduces a linearization of **HO**π, that we here dub **LHO**π. The grammars of processes and values are as follows:

$$\begin{aligned} P &::= \mathbf{0} \mid P \,||\, P \mid a(x).P \mid a(!x).P \mid \overline{a}\langle V\rangle.P \mid (\nu a)P \mid VV; \\ V &::= \star \mid x \mid \lambda x.P \mid \lambda !x.P \mid !V. \end{aligned}$$

On top of the grammar, we must enforce the linearity constraints, which are expressed by the rules in Figure 2. They prove judgements in the form $\Gamma \vdash_{\mathsf{P}} P$ and $\Gamma \vdash_{\mathsf{V}} V$, where Γ is a *context* consisting of a finite set of variables — a single variable may appear in Γ either as x or as $!x$, but not both. Examples of contexts are $x, !y$; or x, y, z; or the empty context $\emptyset$. As usual, we write $!\Gamma$ when all variables of the context (if any) are !-marked. A process P (respectively, a value V) is *well-formed* iff there is a context Γ such that $\Gamma \vdash_{\mathsf{P}} P$ (respectively, $\Gamma \vdash_{\mathsf{V}} V$). In the rules with two premises, observe the implicit contractions on !-marked variables in the context — they allow for transparent duplication. The *depth* of a (occurrence of a) variable x in a process or value is the number of instances of the ! operator it is enclosed to. As an example, if $P = (!x)(y)$, then x has depth 1, while y has depth 0.

A judgement $\Gamma \vdash_{\mathsf{P}} P$ can informally be interpreted as follows. Any variable appearing as x in Γ must occur free exactly once in P; moreover the only occurrence of x is at depth 0 in P (that is, it is not in the scope of any !). On the other hand, any variable y appearing as $!y$ in Γ may occur free any number of times in P, at any depth. Variables like x are *linear*, while those like y are *nonlinear*. Nonlinear variables may only be bound by nonlinear binders (which have a ! to recall this fact).

The operational semantics of **LHO**π is a slight variation on the one of **HO**π, and can be found in Figure 3. The two versions of communication and abstraction (i.e., the linear and the nonlinear one) are governed by two distinct rules. In the nonlinear case the argument to the function (or the value sent through a channel) must be in the correct duplicable form $!V$. Well-formation is preserved by reduction:

Lemma 1 (Subject Reduction) *If* $\vdash_{\mathsf{P}} P$ *and* $P \to_{\mathsf{L}} Q$*, then* $\vdash_{\mathsf{P}} Q$*.*

$$\frac{}{\overline{a}\langle V\rangle.P \mid\mid a(x).Q \to_{\mathsf{L}} P \mid\mid Q[x/V]} \qquad \frac{}{(\lambda x.P)V \to_{\mathsf{L}} P[x/V]}$$

$$\frac{}{\overline{a}\langle !V\rangle.P \mid\mid a(!x).Q \to_{\mathsf{L}} P \mid\mid Q[x/V]} \qquad \frac{}{(\lambda !x.P)!V \to_{\mathsf{L}} P[x/V]}$$

$$\frac{P \to_{\mathsf{L}} Q}{P \mid\mid R \to_{\mathsf{L}} Q \mid\mid R} \qquad \frac{P \to_{\mathsf{L}} Q}{(\nu a)P \to_{\mathsf{L}} (\nu a)Q} \qquad \frac{P \equiv Q \quad Q \to_{\mathsf{L}} R \quad R \equiv S}{P \to_{\mathsf{L}} S}$$

Figure 3: The operational semantics of **LHOπ** processes.

3.1 Embedding Processes into Linear Processes

Processes (and values) can be embedded into linear processes (and values) as follows:

$$\begin{aligned} [\star]_{\mathsf{V}} &= \star; & [\lambda x.Q]_{\mathsf{V}} &= \lambda !x.[P]_{\mathsf{P}};\\ [\mathbf{0}]_{\mathsf{P}} &= \mathbf{0}; & [x]_{\mathsf{V}} &= x;\\ [P \mid\mid Q]_{\mathsf{P}} &= [P]_{\mathsf{P}} \mid\mid [Q]_{\mathsf{P}}; & [a(x).P]_{\mathsf{P}} &= a(!x).[P]_{\mathsf{P}};\\ [\overline{a}\langle V\rangle.P]_{\mathsf{P}} &= \overline{a}\langle ![V]_{\mathsf{V}}\rangle.[P]_{\mathsf{P}}; & [(\nu a)P]_{\mathsf{P}} &= (\nu a)[P]_{\mathsf{P}};\\ [VW]_{\mathsf{P}} &= [V]_{\mathsf{V}}![W]_{\mathsf{V}}. \end{aligned}$$

Linear abstractions and linear inputs never appear in processes obtained via $[\cdot]_{\mathsf{P}}$: whenever a value is sent through a channel or passed to a function, it is made duplicable. The embedding induces a simulation of processes by linear processes:

Proposition 1 (Simulation) *For every process P, $[P]_{\mathsf{P}}$ is well-formed. Moreover, if $P \to_{\mathsf{P}} Q$, then $[P]_{\mathsf{P}} \to_{\mathsf{L}} [Q]_{\mathsf{P}}$.*

By applying the map $[\cdot]_{\mathsf{P}}$ to our example process, *SERVER*, a linear process $SERVER_!$ can be obtained:

$$\begin{aligned} SERVER_! &= (\nu a)(COMP_!(!\star) \mid\mid \overline{a}\langle !COMP_!\rangle);\\ COMP_! &= \lambda !z.(a(!x).(b(!y).\overline{c}\langle !y\rangle.\overline{a}\langle !x\rangle \mid\mid x(!\star))). \end{aligned}$$

4 Termination in Bounded Time: Soft Processes

In view of Proposition 1, **LHOπ** admits non terminating processes. Indeed, the prototypical divergent process from Section 2 can be translated into a linear process:

$$OMEGA_! = (\nu a)((DELTA_!(!\star)) \mid\mid \overline{a}\langle !DELTA_!\rangle), \qquad \text{where} \qquad DELTA_! = \lambda !y.(a(!x).(x\,(!\star) \mid\mid \overline{a}\langle !x\rangle)).$$

$OMEGA_!$ cannot be terminating, since *OMEGA* itself does not terminate.

The more expressive syntax, however, may reveal *why* a process does not terminate. If we trace its execution, we see that the divergence of $OMEGA_!$ comes from $DELTA_!$, where x appears free twice in the inner body $(x\,(!\star) \mid\mid \overline{a}\langle !x\rangle)$: once in the scope of the ! operator, once outside any !. When a value is substituted for x (and thus duplicated) one of the two copies interacts with the other, being copied again. It is this cyclic phenomenon (called *modal impredicativity* in [11]) that is responsible for nontermination.

$$\frac{}{\#\Gamma \vdash_{\mathsf{SP}} \mathbf{0}} \qquad \frac{\Gamma, \#\Lambda \vdash_{\mathsf{SP}} P \quad \Delta, \#\Lambda \vdash_{\mathsf{SP}} Q}{\Gamma, \Delta, \#\Lambda \vdash_{\mathsf{SP}} P \,||\, Q} \qquad \frac{\Gamma, x \vdash_{\mathsf{SP}} P}{\Gamma \vdash_{\mathsf{SP}} a(x).P}$$

$$\frac{\Gamma, !x \vdash_{\mathsf{SP}} P}{\Gamma \vdash_{\mathsf{SP}} a(!x).P} \qquad \frac{\Gamma, \#x \vdash_{\mathsf{SP}} P}{\Gamma \vdash_{\mathsf{SP}} a(!x).P} \qquad \frac{\Gamma, \#\Lambda \vdash_{\mathsf{SV}} V \quad \Delta, \#\Lambda \vdash_{\mathsf{SP}} P}{\Gamma, \Delta, \#\Lambda \vdash_{\mathsf{SP}} \overline{a}\langle V \rangle.P}$$

$$\frac{\Gamma \vdash_{\mathsf{SP}} P}{\Gamma \vdash_{\mathsf{SP}} (\nu a)P} \qquad \frac{\Gamma, \#\Lambda \vdash_{\mathsf{SV}} V \quad \Delta, \#\Lambda \vdash_{\mathsf{SV}} W}{\Gamma, \Delta, \#\Lambda \vdash_{\mathsf{SP}} VW} \qquad \frac{}{\#\Gamma \vdash_{\mathsf{SV}} \star}$$

$$\frac{}{\#\Gamma, x \vdash_{\mathsf{SV}} x} \qquad \frac{}{\#\Gamma, \#x \vdash_{\mathsf{SV}} x} \qquad \frac{\Gamma, x \vdash_{\mathsf{SP}} P}{\Gamma \vdash_{\mathsf{SV}} \lambda x.P}$$

$$\frac{\Gamma, \#x \vdash_{\mathsf{SP}} P}{\Gamma \vdash_{\mathsf{SV}} \lambda !x.P} \qquad \frac{\Gamma, !x \vdash_{\mathsf{SP}} P}{\Gamma \vdash_{\mathsf{SV}} \lambda !x.P} \qquad \frac{\Gamma \vdash_{\mathsf{SV}} V}{!\Gamma, \#\Delta \vdash_{\mathsf{SV}} !V}$$

Figure 4: Processes and values in **SHO**π.

The Linear Logic community has studied in depth the impact of unbalanced and multiple boxes on the complexity of computation, and singled out several (different) sufficient conditions for ensuring not only termination, but termination with prescribed bounds. We will adopt here the conditions arising from Lafont's analysis (and formalized in Soft Linear Logic, **SLL** [10]), leaving to further work the usage of other criteria. We thus introduce the calculus **SHO**π of *soft processes*, for which we will prove termination in polynomial time. In our view, this is the main contribution of the paper.

Soft processes share the same grammar and operational semantics than linear processes (Section 3), but are subjected to stronger constraints, expressed by the well-formation rules of Figure 4. A context Γ can now contain a variable x in at most one of *three* different forms: x, $!x$, or $\#x$. The implicit contraction (or weakening) happens on #-marked variables, but none of them may ever appear inside a !-box. In the last rule it is implicitly assumed that the context Γ in the premise is composed only of linear variables, if any (otherwise the context $!\Gamma$ of the conclusion would be ill-formed). Indeed, the rules amount to say that, if $\Gamma \vdash_{\mathsf{SP}} P$ (and similarly for values), then: (i) any linear variable x in Γ occurs exactly once in P, and at depth 0 (this is as in **LHO**π); (ii) any nonlinear variable $!x$ occurs exactly once in P, and at depth 1; (iii) any nonlinear variable $\#x$ may occur any number of times in P, all of its occurrences must be at level 0. As a result, any bound variable appears in the scope of the binder always at a same level. As in **LHO**π, well-formed processes are closed by reduction:

Proposition 2 *If* $\vdash_{\mathsf{SP}} P$ *and* $P \to_{\mathsf{L}} Q$*, then* $\vdash_{\mathsf{SP}} Q$*.*

The nonterminating process $\mathit{OMEGA}_!$ which started this section is *not* a soft process, because the bound variable x appears twice, once at depth 0 and once depth 1. And this is good news: we would like **SHO**π to be a calculus of terminating processes, at least! But this has some drawbacks: also $\mathit{SERVER}_!$ is not a soft process. Indeed, **SHO**π is not able to discriminate between $\mathit{SERVER}_!$ and $\mathit{OMEGA}_!$, which share a very similar structure. We will come back to this after we proved our main result on the polynomial bound on reduction sequences for soft processes.

4.1 Feasible Termination

This section is devoted to the proof of feasible termination for soft processes. We prove that the length of any reduction sequence from a soft process P is bounded by a polynomial on the size of P. Moreover, the size of any process along the reduction is itself polynomially bounded.

The proof proceeds similarly to the one for **SLL** proof-nets by Lafont [10]. The idea is relatively simple: a weight is assigned to every process and is proved to decrease at any normalization step. The weight of a process can be proved to be an *upper bound* on the size of the process. Finally, a polynomial bound on the weight of a process holds. Altogether, this implies feasible termination.

Before embarking on the proofs, we need some preliminary definitions. First of all, the *size* of a process P (respectively, a value V) is defined simply as the number of symbols in it and is denoted as $|P|$ (respectively, $|V|$) Another crucial attribute of processes and values is their *box depth*, namely the maximum nesting of ! operators inside them; for a process P and a value V, it is denoted either as $\mathbb{B}(P)$ or as $\mathbb{B}(V)$. The *duplicability factor* $\mathbb{D}(P)$ of a process P is the maximum number of free occurrences of a variable x for every binder in P; similarly for values. The precise definition follows, where $\mathbb{FO}(x,P)$ denotes the number of free occurrences on x in P.

$$
\begin{aligned}
\mathbb{D}(\star) = \mathbb{D}(x) = \mathbb{D}(\mathbf{0}) &= 1; & \mathbb{D}(\lambda x.P) = \mathbb{D}(\lambda !x.P) &= \max\{\mathbb{D}(P), \mathbb{FO}(x,P)\};\\
\mathbb{D}(!V) &= \mathbb{D}(V); & \mathbb{D}(P \mid\mid Q) &= \max\{\mathbb{D}(P), \mathbb{D}(Q)\};\\
\mathbb{D}(a(x).P) = \mathbb{D}(a(!x).P) &= \max\{\mathbb{D}(P), \mathbb{FO}(x,P)\}; & \mathbb{D}(\overline{a}\langle V\rangle.P) &= \max\{\mathbb{D}(V), \mathbb{D}(P)\};\\
\mathbb{D}((\nu a)P) &= \mathbb{D}(P); & \mathbb{D}(VW) &= \max\{\mathbb{D}(V), \mathbb{D}(W)\}.
\end{aligned}
$$

Finally, we can define the weight of processes and values. A notion of weight parametrized on a natural number n can be given as follows, by induction on the structure of processes and values:

$$
\begin{aligned}
\mathbb{W}_n(\star) = \mathbb{W}_n(x) = \mathbb{W}_n(\mathbf{0}) &= 1; & \mathbb{W}_n(\lambda x.P) = \mathbb{W}_n(\lambda !x.P) &= \mathbb{W}_n(P);\\
\mathbb{W}_n(!V) &= n \cdot \mathbb{W}_n(V) + 1; & \mathbb{W}_n(P \mid\mid Q) &= \mathbb{W}_n(P) + \mathbb{W}_n(Q) + 1;\\
\mathbb{W}_n(a(x).P) = \mathbb{W}_n(a(!x).P) &= \mathbb{W}_n(P) + 1; & \mathbb{W}_n(\overline{a}\langle V\rangle.P) &= \mathbb{W}_n(V) + \mathbb{W}_n(P);\\
\mathbb{W}_n((\nu a)P) &= \mathbb{W}_n(P); & \mathbb{W}_n(VW) &= \mathbb{W}_n(V) + \mathbb{W}_n(W) + 1.
\end{aligned}
$$

Now, the *weight* $\mathbb{W}(P)$ of a process P is $\mathbb{W}_{\mathbb{D}(P)}(P)$. Similarly for values.

The first auxiliary result is about structural congruence. As one would expect, two structurally congruent terms have identical size, box depth, duplicability factor and weight:

Proposition 3 *if $P \equiv Q$, then $|P| = |Q|$, $\mathbb{B}(P) = \mathbb{B}(Q)$, $\mathbb{D}(P) = \mathbb{D}(Q)$. Moreover, for every n, $\mathbb{W}_n(P) = \mathbb{W}_n(Q)$.*

Observe that Proposition 3 would not hold in presence of structural congruence rules like $P \mid\mid \mathbf{0} \equiv P$ and $(\nu a)\mathbf{0} \equiv a$.

How does $\mathbb{D}(P)$ evolve during reduction? Actually, it cannot grow:

Lemma 2 *If $\vdash_{\mathsf{SP}} Q$ and $Q \rightarrow_{\mathsf{L}} P$, then $\mathbb{D}(Q) \geq \mathbb{D}(P)$.*

Proof. As an auxiliary lemma, we can prove that whenever $\Gamma \vdash_{\mathsf{SP}} P$ and $\emptyset \vdash_{\mathsf{SV}} V, \Delta \vdash_{\mathsf{SV}} W$, both $\mathbb{D}(P[x/V]) \leq \max\{\mathbb{D}(P), \mathbb{D}(V)\}$ and $\mathbb{D}(W[x/V]) \leq \max\{\mathbb{D}(W), \mathbb{D}(V)\}$. This is an easy induction on derivations for $\Gamma \vdash_{\mathsf{SP}} P$ and $\Delta \vdash_{\mathsf{SV}} W$. The thesis follows. □

The weight of a process is an upper bound to the size of the process itself. This means that bounding the weight of a process implies bounding its size. Moreover, the weight of a process strictly decreases at any reduction step.

Lemma 3 *For every P, $\mathbb{W}(P) \geq |P|$.*

Proof. By induction on P, strengthening the induction hypothesis with a similar statement for values. In the induction, observe that $\mathbb{D}(P), \mathbb{D}(V) \geq 1$ for every process P and value V. □

Proposition 4 *If $\vdash_{\mathsf{SP}} Q$ and $Q \rightarrow_{\mathsf{L}} P$, then $\mathbb{W}(Q) > \mathbb{W}(P)$.*

Proof. As an auxiliary result, we need to prove the following (slightly modifications of) substitution lemmas (let $\emptyset \vdash_{\mathsf{SV}} V$ and $n \geq m \geq 1$):

- If $\pi : \Gamma, x \vdash_{\mathsf{SP}} R$, then $\mathbb{W}_m(R[x/V]) \leq \mathbb{W}_n(R) + \mathbb{W}_n(V)$;
- If $\pi : \Gamma, x \vdash_{\mathsf{SV}} W$, then $\mathbb{W}_m(W[x/V]) \leq \mathbb{W}_n(W) + \mathbb{W}_n(V)$;
- If $\pi : \Gamma, \#x \vdash_{\mathsf{SP}} R$, then $\mathbb{W}_m(R[x/V]) \leq \mathbb{W}_n(R) + \mathbb{FO}(x,R) \cdot \mathbb{W}_n(V)$;
- If $\pi : \Gamma, \#x \vdash_{\mathsf{SV}} W$, then $\mathbb{W}_m(W[x/V]) \leq \mathbb{W}_n(W) + \mathbb{FO}(x,W) \cdot \mathbb{W}_n(V)$;
- If $\pi : \Gamma, !x \vdash_{\mathsf{SP}} R$, then $\mathbb{W}_m(R[x/V]) \leq \mathbb{W}_n(R) + n \cdot \mathbb{W}_n(V)$;
- If $\pi : \Gamma, !x \vdash_{\mathsf{SV}} W$, then $\mathbb{W}_m(W[x/V]) \leq \mathbb{W}_n(W) + n \cdot \mathbb{W}_n(V)$;

This is an induction on π. An inductive case:

- If π is:

$$\frac{\Gamma, x \vdash_{\mathsf{SV}} Z}{!\Gamma, !x, \#\Delta \vdash_{\mathsf{SV}} !Z}$$

  then $W = !Z$ and $(!Z)[x/V]$ is simply $!(Z[x/V])$. As a consequence:

$$\begin{aligned}\mathbb{W}_m(W[x/V]) &= m \cdot \mathbb{W}_m(Z[x/V]) + 1 \leq n \cdot (\mathbb{W}_n(Z) + \mathbb{W}_n(V)) + 1 = n \cdot \mathbb{W}_n(Z) + n \cdot \mathbb{W}_n(V) + 1 \\ &= \mathbb{W}_n(!Z) + n \cdot \mathbb{W}_n(V) = \mathbb{W}_n(W) + n \cdot \mathbb{W}_n(V).\end{aligned}$$

With the above observations in hand, we can easily prove the thesis by induction on any derivation ρ of $P \rightarrow_{\mathsf{P}} Q$:

- Suppose ρ is

$$\overline{\overline{a}\langle V \rangle.R \,||\, a(x).S \rightarrow_{\mathsf{L}} R \,||\, S[x/V]}$$

  From $\emptyset \vdash_{\mathsf{SP}} \overline{a}\langle V \rangle.R \,||\, a(x).S$, it follows that $\emptyset \vdash_{\mathsf{SP}} R$, $\emptyset \vdash_{\mathsf{SV}} V$ and $x \vdash_{\mathsf{SP}} S$. As a consequence, since $\mathbb{D}(Q) \leq \mathbb{D}(P)$,

$$\begin{aligned}\mathbb{W}(P) &= \mathbb{W}(\overline{a}\langle V \rangle.R \,||\, a(x).S) = \mathbb{W}_{\mathbb{D}(P)}(V) + \mathbb{W}_{\mathbb{D}(P)}(R) + \mathbb{W}_{\mathbb{D}(P)}(S) + 2 \\ &\geq \mathbb{W}_{\mathbb{D}(Q)}(S[x/V]) + \mathbb{W}_{\mathbb{D}(Q)}(R) + 2 > \mathbb{W}_{\mathbb{D}(Q)}(S[x/V]) + \mathbb{W}_{\mathbb{D}(Q)}(R) + 1 = \mathbb{W}_{\mathbb{D}(Q)}(S[x/V] \,||\, R).\end{aligned}$$

- Suppose ρ is

$$\overline{\overline{a}\langle !V \rangle.R \,||\, a(!x).S \rightarrow_{\mathsf{L}} R \,||\, S[x/V]}$$

  From $\emptyset \vdash_{\mathsf{SP}} \overline{a}\langle V \rangle.R \,||\, a(x).S$, it follows that $\emptyset \vdash_{\mathsf{SP}} R$, $\emptyset \vdash_{\mathsf{SV}} V$ and either $!x \vdash_{\mathsf{SP}} S$ or $\#x \vdash_{\mathsf{SP}} S$. In the first case:

$$\begin{aligned}\mathbb{W}(P) &= \mathbb{W}(\overline{a}\langle !V \rangle.R \,||\, a(x).S) = \mathbb{W}_{\mathbb{D}(P)}(!V) + \mathbb{W}_{\mathbb{D}(P)}(R) + \mathbb{W}_{\mathbb{D}(P)}(S) + 2 \\ &= \mathbb{D}(P) \cdot \mathbb{W}_{\mathbb{D}(P)}(V) + \mathbb{W}_{\mathbb{D}(P)}(R) + \mathbb{W}_{\mathbb{D}(P)}(S) + 3 \geq \mathbb{W}_{\mathbb{D}(Q)}(S[x/V]) + \mathbb{W}_{\mathbb{D}(Q)}(R) + 3 \\ &> \mathbb{W}_{\mathbb{D}(Q)}(S[x/V]) + \mathbb{W}_{\mathbb{D}(Q)}(R) + 1 = \mathbb{W}_{\mathbb{D}(Q)}(S[x/V] \,||\, R).\end{aligned}$$

  In the second case:

$$\begin{aligned}\mathbb{W}(P) &= \mathbb{W}(\overline{a}\langle !V \rangle.R \,||\, a(x).S) = \mathbb{W}_{\mathbb{D}(P)}(!V) + \mathbb{W}_{\mathbb{D}(P)}(R) + \mathbb{W}_{\mathbb{D}(P)}(S) + 2 \\ &= \mathbb{D}(P) \cdot \mathbb{W}_{\mathbb{D}(P)}(V) + \mathbb{W}_{\mathbb{D}(P)}(R) + \mathbb{W}_{\mathbb{D}(P)}(S) + 3 \\ &\geq \mathbb{FO}(x,S) \cdot \mathbb{W}_{\mathbb{D}(P)}(V) + \mathbb{W}_{\mathbb{D}(P)}(R) + \mathbb{W}_{\mathbb{D}(P)}(S) + 3 \\ &\geq \mathbb{W}_{\mathbb{D}(Q)}(S[x/V]) + \mathbb{W}_{\mathbb{D}(Q)}(R) + 3 > \mathbb{W}_{\mathbb{D}(Q)}(S[x/V]) + \mathbb{W}_{\mathbb{D}(Q)}(R) + 1 = \mathbb{W}_{\mathbb{D}(Q)}(S[x/V] \,||\, R).\end{aligned}$$

- Suppose ρ is

$$\frac{\sigma : R \rightarrow_{\mathsf{L}} S}{R \mid\mid T \rightarrow_{\mathsf{L}} S \mid\mid T}$$

From $\emptyset \vdash_{\mathsf{SP}} R \mid\mid T$, it follows that $\emptyset \vdash_{\mathsf{SP}} R$ and $\emptyset \vdash_{\mathsf{SP}} T$. By induction hypothesis on σ, this yields $\mathbb{W}(R) > \mathbb{W}(S)$, and in turn $\mathbb{W}(R) = \mathbb{W}(R) + \mathbb{W}(T) + 1 > \mathbb{W}(S) + \mathbb{W}(T) + 1 = \mathbb{W}(S)$.

This concludes the proof. □

Lemma 3 and Proposition 4 together imply that the weight is an upper bound to both the number of reduction steps a process can perform and the size of any reduct. So, the only missing tale is bounding the weight itself:

Proposition 5 *For every process P, $\mathbb{W}(P) \leq |P|^{\mathbb{B}(P)+1}$.*

Proof. By induction on P, enriching the thesis with an analogous statement for values: $\mathbb{W}(V) \leq |V|^{\mathbb{B}(V)+1}$.
□

Putting all the ingredients together, we reach our soundness result with respect polynomial time:

Theorem 1 *There is a family of polynomials $\{p_n\}_n$ such that for every process P and for every m, if $P \rightarrow_{\mathsf{L}}^{m} Q$, then $m, |Q| \leq p_{\mathbb{B}(P)}(|P|)$.*

The polynomials in Theorem 1 depend on terms, so the bound on the number of internal actions is not polynomial, strictly speaking. Please observe, however, that all processes with the same box depth b are governed by the same polynomial p_b, similarly to what happens in Soft Linear Logic.

4.2 Completeness?

Soundness of a formal system with respect to some semantic criterion is useless unless one shows that the system is also *expressive enough*. In implicit computational complexity, programming languages are usually proved both sound and *extensionally complete* with respect to a complexity class. Not only any program can be normalized in bounded time, but every function in the class can be computed by a program in the system. Preliminary to any completeness result for **SHO**π, however, would be the definition of what a complexity class for processes should be (as opposed to the well known definition for functions or problems). This is an elusive—and very interesting—problem that we cannot tackle in this preliminary work and that we leave for future work.

Certainly the expressiveness of **SHO**π is weak if we take into account the visible actions of the processes (i.e., their interactions with the environment). This is due to the limited possibilities of copying, and hence also of writing recursive process behaviours. Indeed, one cannot consider **SHO**π, on its own, as a general-purpose calculus for concurrency. However, we believe that the study of **SHO**π, or similar languages, could be fruitful in establishing bounds on the internal behaviour of parts, or components, of a concurrent systems; for instance, on the time and space that a process may take to answer a query from another process (in this case the **SHO**π techniques would be applied to the parts of the syntax of the process that describe its internal computation after the query). Next section considers a possible direction of development of **SHO**π, allowing more freedom on the external actions of the processes.

We are convinced, on the other hand, that a minimal completeness result can be given, namely the possibility of representing all polynomial time *functions* (or problems) in **SHO**π. Possibly, this could be done by encoding Soft Linear Logic into **SHO**π through a continuation-passing style translation. We leave this to future work.

5 An Extension to SHOπ: Spawning

In this section we propose an extension of **SHO**π that allows us to accept processes such as $SERVER_!$, capable of performing infinitely many interactions with their external environment while maintaining polynomial bounds on the number of internal steps they can make between any two external actions.

The reason why $SERVER_!$ is *not* a **SHO**π process has to do with the bound variable x in the subprocess $COMP_!$:

$$COMP_! = \lambda !z.(a(!x).(b(!y).\overline{c}\langle !y\rangle.x(!\star) \mid\mid \overline{a}\langle !x\rangle)),$$

The variable appears twice in the body $(b(!y).\overline{c}\langle !y\rangle.x(!\star) \mid\mid \overline{a}\langle !x\rangle)$, at two different !-depths. This pattern is not permitted in **SHO**π, because otherwise also the nonterminating process $OMEGA_!$ would be in the calculus. There is however a major difference between $OMEGA_!$ and $SERVER_!$: in $COMP_!$, one of the two occurrences of x (the one at depth 0) is part of the continuation of an input on b; moreover, such channel b is only used by $SERVER_!$ in input — $SERVER_!$ does not own the output capability. This implies that whatever process will substitute that occurrence of x, it will be able to interact with the environment only *after* an input on b is performed. So, its "computational weight" does not affect the number of reduction steps made by the process *before* such an input occurs. This phenomenon, which does not occur in $OMEGA_!$, can be seen as a form of process spawning: $COMP_!$ can be copied an unbounded number of times, but the rhythm of the copying is dictated by the input actions at b.

Consider a subset $\mathscr{IC}$ of $\mathscr{C}$ (where $\mathscr{C}$ is the set of all channels which can appear in processes). The process calculus **EHO**$\pi(\mathscr{IC})$ is an extension of **SHO**π parametrized on $\mathscr{IC}$. What **EHO**$\pi(\mathscr{IC})$ adds to **SHO**π is precisely the possibility of marking a subprocess as a component which can be spawned. This is accomplished with a new operator $\square$. Channels in $\mathscr{IC}$ are called *input channels*, because outputs are forbidden on them. The syntax of processes and values is enriched as follows:

$$P ::= \ldots \mid a(\square x).P;$$
$$V ::= \ldots \mid \lambda\square x.P \mid \square V;$$

but outputs can only be performed on channels not in $\mathscr{IC}$. The term $\square V$ is a value (i.e., a parametrized process) which can be spawned. Spawning itself is performed by passing a process $\square V$ to either an abstraction $\lambda\square x.P$ or an input $a(\square x).P$. In both cases, exactly one occurrence of x in P is the scope of a $\square$ operator, and only one of the following two conditions holds:

1. The occurrence of x in the scope of a $\square$ operator is part of the continuation of an input channel a, and all other occurrences of x in P are at depth 0.
2. There are no other occurrences of x in P.

The foregoing constraints are enforced by the well-formation rules in Figure 5. The well-formation rules of **EHO**$\pi(\mathscr{IC})$ are considerably more complex than the ones of **SHO**π. Judgements have the form $\Gamma \vdash_{\mathsf{EP}} P$ or $\Gamma \vdash_{\mathsf{EV}} V$, where a variable x can occur in Γ in one of five different forms:

- As either x, $!x$ or $\#x$: here the meaning is exactly the one from **SHO**π (see Section 4).
- As $\square x$: the variable x then appears exactly once in P, in the scope of a spawning operator $\square$.
- As $\diamond x$: x occurs at least once in P, once in the scope of a $\square$ operator (itself part of the continuation for an input channel), and possibly many times at depth 0.

A variable marked as $\diamond x$ can "absorb" the same variable declared as $\#x$ in binary well-formation rules (i.e. the ones for applications, outputs, etc.). Note the special well-formation rules that are only applicable with an input channel: in that case a portion of the context $\square\Delta$ becomes $\diamond\Delta$.

The operational semantics is obtained adding to Figure 3 the following two rules:

$$\frac{}{\overline{a}\langle\square R\rangle.P \mid\mid a(\square x).Q \rightarrow_{\mathsf{L}} P \mid\mid Q[x/R]} \qquad \frac{}{(\lambda\square x.P)\square Q \rightarrow_{\mathsf{L}} P[x/Q]}$$

$$
\frac{}{\#\Gamma \vdash_{\mathsf{EP}} \mathbf{0}}
\qquad
\frac{\Gamma, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EP}} P \quad \Delta, \#\Lambda, \#\Theta \vdash_{\mathsf{EP}} Q}{\Gamma, \Delta, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EP}} P \,||\, Q}
\qquad
\frac{\Gamma, x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EP}} a(x).P}
$$

$$
\frac{\Gamma, !x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EP}} a(!x).P}
\qquad
\frac{\Gamma, \#x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EP}} a(!x).P}
\qquad
\frac{\Gamma, \Box x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EP}} a(\Box x).P}
\qquad
\frac{\Gamma, \Diamond x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EP}} a(\Box x).P}
$$

$$
\frac{\Gamma, \Box\Delta, x \vdash_{\mathsf{EP}} P \quad a \in \mathscr{IC}}{\Gamma, \Diamond\Delta \vdash_{\mathsf{EP}} a(x).P}
\qquad
\frac{\Gamma, \Box\Delta, !x \vdash_{\mathsf{EP}} P \quad a \in \mathscr{IC}}{\Gamma, \Diamond\Delta \vdash_{\mathsf{EP}} a(!x).P}
$$

$$
\frac{\Gamma, \Box\Delta, \#x \vdash_{\mathsf{EP}} P \quad a \in \mathscr{IC}}{\Gamma, \Diamond\Delta \vdash_{\mathsf{EP}} a(!x).P}
\qquad
\frac{\Gamma, \Box\Delta, \Box x \vdash_{\mathsf{EP}} P \quad a \in \mathscr{IC}}{\Gamma, \Diamond\Delta \vdash_{\mathsf{EP}} a(\Box x).P}
\qquad
\frac{\Gamma, \Box\Delta, \Diamond x \vdash_{\mathsf{EP}} P \quad a \in \mathscr{IC}}{\Gamma, \Diamond\Delta \vdash_{\mathsf{EP}} a(\Box x).P}
$$

$$
\frac{\Gamma, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EV}} V \quad \Delta, \#\Lambda, \#\Theta \vdash_{\mathsf{EP}} P}{\Gamma, \Delta, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EP}} \overline{a}\langle V \rangle.P}
\qquad
\frac{\Gamma, \#\Lambda, \#\Theta \vdash_{\mathsf{EV}} V \quad \Delta, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EP}} P}{\Gamma, \Delta, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EP}} \overline{a}\langle V \rangle.P}
$$

$$
\frac{\Gamma \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EP}} (\nu a)P}
\qquad
\frac{\Gamma, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EV}} V \quad \Delta, \#\Lambda, \#\Theta \vdash_{\mathsf{EV}} W}{\Gamma, \Delta, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EP}} VW}
\qquad
\frac{\Gamma, \#\Lambda, \#\Theta \vdash_{\mathsf{EV}} V \quad \Delta, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EV}} W}{\Gamma, \Delta, \#\Lambda, \Diamond\Theta \vdash_{\mathsf{EP}} VW}
$$

$$
\frac{}{\#\Gamma \vdash_{\mathsf{EV}} \star}
\qquad
\frac{}{\#\Gamma, x \vdash_{\mathsf{EV}} x}
\qquad
\frac{}{\#\Gamma, \#x \vdash_{\mathsf{EV}} x}
\qquad
\frac{\Gamma, x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EV}} \lambda x.P}
$$

$$
\frac{\Gamma, \#x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EV}} \lambda !x.P}
\qquad
\frac{\Gamma, !x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EV}} \lambda !x.P}
\qquad
\frac{\Gamma, \Box x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EV}} \lambda \Box x.P}
$$

$$
\frac{\Gamma, \Diamond x \vdash_{\mathsf{EP}} P}{\Gamma \vdash_{\mathsf{EV}} \lambda \Box x.P}
\qquad
\frac{\Gamma \vdash_{\mathsf{EV}} V}{!\Gamma, \#\Delta \vdash_{\mathsf{EV}} !V}
\qquad
\frac{\Gamma \vdash_{\mathsf{EV}} V}{\Box\Gamma, \#\Delta \vdash_{\mathsf{EV}} \Box V}
$$

Figure 5: Processes and values in **EHO**$\pi(\mathscr{IC})$.

As expected,

Lemma 4 (Subject Reduction) *If* $\vdash_{\mathsf{EP}} P$ *and* $P \rightarrow_{\mathsf{L}} Q$*, then* $\vdash_{\mathsf{EP}} Q$*.*

The process $SERVER_!$ is a **EHO**$\pi(\mathscr{IC})$ process once $COMP_!$ is considered as a spawned process and $b \in \mathscr{IC}$: define

$$
\begin{aligned}
SERVER_\Box &= (\nu a)(COMP_\Box(!\star) \,||\, \overline{a}\langle \Box COMP_\Box \rangle);\\
COMP_\Box &= \lambda !z.(a(\Box x).(b(!y).\overline{c}\langle !y \rangle.\overline{a}\langle \Box x \rangle \,||\, x(!\star))).
\end{aligned}
$$

and consider the following derivations:

$$
\cfrac{\cfrac{\cfrac{\emptyset \vdash_{\mathsf{EV}} COMP_\Box \quad \cfrac{\emptyset \vdash_{\mathsf{EV}} \star}{\emptyset \vdash_{\mathsf{EV}} !\star}}{\emptyset \vdash_{\mathsf{EP}} COMP_\Box(!\star)} \quad \cfrac{\cfrac{\emptyset \vdash_{\mathsf{EV}} COMP_\Box}{\emptyset \vdash_{\mathsf{EV}} \Box COMP_\Box}}{\emptyset \vdash_{\mathsf{EP}} \overline{a}\langle \Box COMP_\Box \rangle}}{\emptyset \vdash_{\mathsf{EP}} COMP_\Box(!\star) \,||\, \overline{a}\langle \Box COMP_\Box \rangle}}{\emptyset \vdash_{\mathsf{EP}} (\nu a)(COMP_\Box(!\star) \,||\, \overline{a}\langle \Box COMP_\Box \rangle)}
$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x \vdash_{\mathsf{EV}} x}}{\#z, \Box x \vdash_{\mathsf{EV}} \Box x}}{\#z, \Box x \vdash_{\mathsf{EP}} \overline{a}\langle \Box x\rangle} \quad \dfrac{\dfrac{}{y \vdash_{\mathsf{EV}} y}}{!y \vdash_{\mathsf{EV}} !y}}{\#z, \Box x, !y \vdash_{\mathsf{EP}} \overline{c}\langle !y\rangle.\overline{a}\langle \Box x\rangle} \quad b \in \mathscr{IC}}{\#z, \Diamond x \vdash_{\mathsf{EP}} b(!y).\overline{c}\langle !y\rangle.\overline{a}\langle \Box x\rangle} \quad \dfrac{\dfrac{}{\#x \vdash_{\mathsf{EV}} x} \quad \dfrac{\emptyset \vdash_{\mathsf{EV}} \star}{\emptyset \vdash_{\mathsf{EV}} !\star}}{\#x \vdash_{\mathsf{EP}} x(!\star)}}{\dfrac{\#z, \Diamond x \vdash_{\mathsf{EP}} b(!y).\overline{c}\langle !y\rangle.\overline{a}\langle \Box x\rangle \parallel x(!\star)}{\dfrac{\#z \vdash_{\mathsf{EP}} a(\Box x).(b(!y).\overline{c}\langle !y\rangle.\overline{a}\langle \Box x\rangle \parallel x(!\star))}{\emptyset \vdash_{\mathsf{EV}} \lambda !z.a(\Box x).(b(!y).\overline{c}\langle !y\rangle.\overline{a}\langle \Box x\rangle \parallel x(!\star))}}}$$

The use in **EHO**$\pi(\mathscr{IC})$ of a distinct set of input channels may still be seen as rigid. For instance, it prevents from accepting *SERVER*$_\Box$ in parallel with a client of the server itself (because the client uses the request channel of the server in output); similarly, it prevents from accepting reentrant servers (servers that can invoke themselves). As pointed out earlier, we are mainly interested in techniques capable of ensuring polynomial bounds on *components* of concurrent systems (so for instance, bounds on the server, rather than on the composition of the server and a client). In any case, this paper represents a preliminary investigation, and further refinements or extensions of **EHO**$\pi(\mathscr{IC})$ may well be possible.

5.1 Feasible Termination

The proof of feasible termination for **EHO**$\pi(\mathscr{IC})$ is similar in structure to the one for **SHO**π (see Section 4.1). However, some additional difficulties due to the presence of spawning arise.

The auxiliary notions we needed in the proof of feasible termination for **SHO**π can be easily extended to **EHO**$\pi(\mathscr{IC})$ as follows: The architecture of the soundness proof is similar to the one for linear processes. The box depth, duplicability factor and weight of a process are defined as for soft processes, plus:

$$\begin{array}{rlrlrl}
\mathbb{B}(\lambda\Box x.P) &= \mathbb{B}(P); & \mathbb{D}(\lambda\Box x.P) &= \max\{\mathbb{D}(P), \mathbb{FO}(x,P)\}; & \mathbb{W}_n(\lambda\Box x.P) &= \mathbb{W}_n(P);\\
\mathbb{B}(\Box V) &= \mathbb{B}(V)+1; & \mathbb{D}(\Box V) &= \mathbb{D}(V); & \mathbb{W}_n(\Box V) &= n\cdot\mathbb{W}_n(V)+1;\\
\mathbb{B}(a(\Box x).P) &= \mathbb{B}(P); & \mathbb{D}(a(\Box x).P) &= \max\{\mathbb{D}(P), \mathbb{FO}(x,P)\}; & \mathbb{W}_n(a(\Box x).P) &= \mathbb{W}_n(P)+1.
\end{array}$$

Informally, the spawning operator $\Box$ acts as ! in all the definitions above. The weight $\mathbb{W}(P)$, still defined as $\mathbb{W}_{\mathbb{D}(P)}(P)$ is again an upper bound to the size of P, but is not guaranteed to decrease at any reduction step. In particular, spawning can make $\mathbb{W}(P)$ bigger. As a consequence, two new auxiliary notions are needed. The first one is similar to the weight of processes and values, but is computed without taking into account whatever happens after an input on a channel $a \in \mathscr{IC}$. It is parametric on a natural number n and is defined as follows:

$$\begin{array}{rlrl}
\mathbb{I}_n(\star) = \mathbb{I}_n(x) = \mathbb{I}_n(\mathbf{0}) &= 1 & \mathbb{I}_n(\lambda x.P) = \mathbb{I}_n(\lambda !x.P) = \mathbb{I}_n(\lambda\Box x.P) &= \mathbb{I}_n(P)\\
\mathbb{I}_n(!V) = \mathbb{I}_n(\Box V) &= n\cdot\mathbb{I}_n(V)+1 & \mathbb{I}_n(P \parallel Q) &= \mathbb{I}_n(P)+\mathbb{I}_n(Q)+1\\
\mathbb{I}_n(a(x).P) = \mathbb{I}_n(a(!x).P) = \mathbb{I}_n(a(\Box x).P) &= \begin{cases} 0 & \text{if } a \in \mathscr{IC}\\ \mathbb{I}_n(P)+1 & \text{otherwise}\end{cases} & \mathbb{I}_n(\overline{a}\langle V\rangle.P) &= \mathbb{I}_n(V)+\mathbb{I}_n(P)\\
\mathbb{I}_n((\nu a)P) &= \mathbb{I}_n(P) & \mathbb{I}_n(PQ) &= \mathbb{I}_n(P)+\mathbb{I}_n(Q)+1
\end{array}$$

The *weight before input* $\mathbb{I}(P)$ of a process P is simply $\mathbb{I}_{\mathbb{D}(P)}(P)$. As we will see, $\mathbb{I}(P)$ *is* guaranteed to decrease at any reduction step, but this time it is not an upper bound to the size of the underlying process. The second auxiliary notion captures the potential growth of processes due to spawning and is again parametric on a natural number n:

$$
\begin{array}{rr}
\mathbb{P}_n(\star) = \mathbb{P}_n(x) = \mathbb{P}_n(\mathbf{0}) = 0 & \mathbb{P}_n(\lambda x.P) = \mathbb{P}_n(\lambda !x.P) = \mathbb{P}_n(\lambda \Box x.P) = \mathbb{P}_n(P) \\
\mathbb{P}_n(!V) = n \cdot \mathbb{P}_n(V) & \mathbb{P}_n(\Box V) = n \cdot \mathbb{P}_n(V) + n \cdot \mathbb{W}_n(V) \\
\mathbb{P}_n(P \mid\mid Q) = \mathbb{P}_n(P) + \mathbb{P}_n(Q) & \mathbb{P}_n(a(x).P) = \mathbb{P}_n(a(!x).P) = \mathbb{P}_n(a(\Box x).P) = \left\{ \begin{array}{ll} 0 & \text{if } a \in \mathscr{IC} \\ \mathbb{P}_n(P) & \text{otherwise} \end{array} \right. \\
\mathbb{P}_n(\overline{a}\langle V \rangle.P) = \mathbb{P}_n(V) + \mathbb{P}_n(P) & \mathbb{P}_n((\nu a)P) = \mathbb{P}_n(P) \\
\mathbb{P}_n(VW) = \mathbb{P}_n(V) + \mathbb{P}_n(W) &
\end{array}
$$

Again, the *potential growth* $\mathbb{P}(P)$ of a process P is $\mathbb{P}_{\mathbb{D}(P)}(P)$. Proposition 3, Lemma 2 and Lemma 3 from Section 4.1 continue to hold for $\mathbf{EHO}\pi(\mathscr{IC})$, and their proofs remain essentially unchanged. Proposition 4 is true only if the weight before input replaces the weight:

Proposition 6 *If $\emptyset \vdash_{\mathsf{SP}} Q$ and $Q \rightarrow_{\mathsf{L}} P$, then $\mathbb{I}(Q) > \mathbb{I}(P)$.*

The potential growth of a process P cannot increase during reduction. Moreover, the weight can increase, but at most by the decrease in the potential growth. Formally:

Proposition 7 *If $\emptyset \vdash_{\mathsf{SP}} Q$ and $Q \rightarrow_{\mathsf{L}} P$, then $\mathbb{P}(Q) \geq \mathbb{P}(P)$ and $\mathbb{W}(Q) + \mathbb{P}(Q) \geq \mathbb{W}(P) + \mathbb{P}(P)$.*

Polynomial bounds on all the attributes of processes we have defined can be proved:

Proposition 8 *For every process P, $\mathbb{W}(P) \leq |P|^{\mathbb{B}(P)+1}$, $\mathbb{I}(P) \leq |P|^{\mathbb{B}(P)+1}$ and $\mathbb{P}(P) \leq \mathbb{B}(P)\mathbb{W}(P)$.*

And, as for $\mathbf{SHO}\pi$, we get a polynomial bound in the number of reduction steps from any process:

Theorem 2 *There is a family of polynomials $\{p_n\}_{n \in \mathbb{N}}$ such that for every process P and for every m, if $P \rightarrow_{\mathsf{L}}^m Q$, then $m, |Q| \leq p_{\mathbb{B}(P)}(|P|)$.*

Proofs for the results above have been elided, due to space constraints. Their structure, however, reflects the corresponding proofs for $\mathbf{SHO}\pi$ (see Section 4.1). As an example, proofs of propositions 6 and 7 are both structured around appropriate substitution lemmas.

6 Conclusions

Goal of this preliminary essay was to verify whether we could apply to process algebras the technologies for resource control that have been developed in the so-called “light logics” and have been successfully applied so far to paradigmatic functional programming. We deliberately adopted a minimalistic approach: applications between processes restricted to values, the simplest available logic, a purely linear language (i.e., no weakening/erasing on non marked formulas), no types, no search for maximal expressivity. In this way the result of the experiment would have had a clear single outcome. We believe this outcome is a clear positive, and that this paper demonstrates it.

Several issues must be investigated further, of course, so that this first experiment may become a solid contribution. First, one may wonder whether other complexity conscious fragments of linear logic can be used in place of **SLL** as guideline for box control. **SLL** is handy as a first try, because of its simplicity, but we do believe that analogous results could be obtained starting from Light Affine Logic, designed by Asperti and Roversi [2] after Girard’s treatment of the purely linear case. This would also allow unrestricted erasing of processes, leaving marked boxes only for duplication. Second, individuate a richer language of processes, still amenable to the soft (or light) treatment. Section 5 suggests a possible direction, but many others are possible. Third, the very interesting problem of studying the notion of complexity class in the process realm.

In the paper, we have proved polynomial bounds for $\mathbf{SHO}\pi$, obtained from the the Higher-Order π-calculus by imposing constraints inspired by Soft Linear Logic. We have then considered an extension of $\mathbf{SHO}\pi$, taking into account features specific to processes, notably the existence of channels: in process calculi a reduction step does not need to be anonymous, as in the λ-calculus, but may result from an interaction along a channel. An objective of the extension was to accept processes that are programmed to have unboundedly many external actions (i.e., interactions with their environment) but that remain polynomial on the internal work performed between any two external activities. Our definition of the extended class, $\mathbf{EHO}\pi(\mathscr{I}\mathscr{C})$, relies on the notion of input channel — a channel that is used in a process only in input. This allows us to have more flexibility in the permitted forms of copying. We have proposed $\mathbf{EHO}\pi(\mathscr{I}\mathscr{C})$ because this class seems mathematically simple and practically interesting. These claims, however, need to be sustained by more evidence. Furher, other refinements of $\mathbf{SHO}\pi$ are possible. Again, more experimentation with examples is needed to understand where to focus attention.

Another question related to the interplay between internal and external actions of processes is whether the polynomial bounds on internal actions change when external actions are performed.

Summarizing, we started with a question ("Can ICC be applied to process algebras?") and ended up with a positive answer and many more different questions. But this is a feature, and not a bug.

References

[1] Roberto M. Amadio & Frédéric Dabrowski (2007): *Feasible Reactivity in a Synchronous π-calculus*. In: *PPDP 2007*, ACM, pp. 221–230.

[2] A. Asperti & L. Roversi (2002): *Intuitionistic Light Affine Logic*. *TOCL* 3(1), pp. 1–39.

[3] Romain Demangeon, Daniel Hirschkoff & Davide Sangiorgi (2010): *Termination in Higher-Order Concurrent Calculi*. In: *FSEN 2009*, *LNCS* 5961, pp. 81–96.

[4] Romain Demangeon, Daniel Hirschkoff & Davide Sangiorgi (2010): *Termination in Impure Concurrent Langages*. In: *CONCUR 2010*, *LNCS* 6269, pp. 328–342.

[5] Thomas Ehrhard & Olivier Laurent (2008): *Acyclic Solos and Differential Interaction Nets*. Technical report PPS Paris VII.

[6] Thomas Ehrhard & Laurent Regnier (2006): *Differential Interaction Nets*. *TCS* 364(2), pp. 166–195.

[7] J.-Y. Girard (1987): *Linear Logic*. *TCS* 50, pp. 1–102.

[8] J.-Y. Girard (1998): *Light Linear Logic*. *I&C* 143(2), pp. 175–204.

[9] Naoki Kobayashi & Davide Sangiorgi (2008): *A Hybrid Type System for Lock-Freedom of Mobile Processes*. In: *CAV 2008*, *LNCS* 5123, pp. 80–93.

[10] Y. Lafont (2004): *Soft Linear Logic and Polynomial Time*. *TCS* 318(1-2), pp. 163–180.

[11] Ugo Dal Lago, Luca Roversi & Luca Vercelli (2009): *Taming Modal Impredicativity: Superlazy Reduction*. In: *LFCS 2009*, *LNCS* 5407, pp. 137–151.

[12] Davide Sangiorgi & David Walker (2001): *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press.

[13] Philip Wadler (1994): *A Syntax for Linear Logic*. In: *MFPS 1993*, *LNCS* 802, pp. 513–529.

[14] N. Yoshida, M. Berger & Honda. K. (2001): *Strong Normalisation in the π-Calculus*. In: *LICS 2001*, IEEE, pp. 311–322.