Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 338 (2018) 97–114

www.elsevier.com/locate/entcs

An Interpretation of CCS into Ludics

Stefano Del Vecchio[1,2] and Virgile Mogbil[1]

[1] *LIPN, CNRS – Université Paris 13, Villetaneuse, France*
[2] *Dipartimento di Matematica e Fisica, Università Roma Tre, Roma, Italia.*

Abstract

Starting from works aimed at extending the Curry-Howard correspondence to process calculi through linear logic, we give another Curry-Howard counterpart for Milner's Calculus of Communicating Systems (CCS) by taking Ludics as the target system. Indeed interaction, Ludics' dynamic, allows to fully represent both the non-determinism and non-confluence of the calculus.
We give an interpretation of CCS processes into carefully defined behaviours of Ludics using a new construction, called *directed behaviour*, that allows controlled interaction paths by using pruned designs. We characterize the execution of processes as interaction on behaviours, by implicitly representing the causal order and conflict relation of event structures. As a direct consequence, we are also able to interpret deadlocked processes, and identify deadlock-free ones.

Keywords: Calculus of Communicating Systems (CCS), Linear logic, Ludics interaction, non-determinism.

1 Introduction

Process algebras are an approach to concurrent theory, to model interactive systems, based on communication, often as message-passing, and reasoning on primitive operators like parallel composition; among the most known and used systems there are Milner's *Calculus of Communicating Systems* (CCS) [14], and the π-calculus ([15], [16]). In our work we focus on the former to give an interpretation in the proof theory setting of *Ludics* [12], which bring back together syntax and semantics following the paradigm of interactive computation, similarly to what is done in game semantics. Finding a proper Curry-Howard counterpart for such calculi could provide a logical foundation to concurrent computation, and some insight into its dynamic.

Usual models for concurrency can be used to give semantics to process algebra, e.g. event structures [23] for CCS [22]. Like Petri Nets [18], event structures are a *true concurrent* model but based on explicit causal order and conflict relations to reveal concurrency. Using closure on these notions, semantical properties easily characterize behaviours such as (no) conflicts, choice independence and confluence. Such closures are also present in our interpretation but internalized in *Ludics' behaviours* directly by bi-orthogonality. From another view point, our Ludics interpretation of

https://doi.org/10.1016/j.entcs.2018.10.007
1571-0661/

CCS expresses all possible schedulings under non-deterministic choices. Somehow this brings it closer to *interleaving* models for concurrency like traces and *labeled transition systems* [24] whose main ability is to characterize observational equivalence.

Motivations and related works. The motivations of this work come from studies aiming at extending the *Curry-Howard* correspondence outside the functional setting, to model concurrency. Many approaches achieved such a correspondence (as [7]), though by imposing determinism to processes, i.e. limited to concurrent systems where only a deterministic behaviour can occur. However, reduction on processes is, in general, non-deterministic and non-confluent: this intrinsically limits a possible correspondence with cut elimination or normalization of proofs, whose nature is confluent and deterministic, effectively restraining a process to functional behaviour.

We start from works based on Linear logic [11] (which inspired many systems, being a logic particularly tied to interaction) that stress the difficulties with the mentioned approaches; in particular [4], [3], [5] and [8]. In [8] the authors show that differential nets are a suitable target system for a correspondence, though the translation proposed is not *modular*, in the sense that we cannot compose two differential nets to represent the (parallel) composition of two processes. But even proof nets, that have similar dynamics to process calculi, with a local and asynchronous cut elimination procedure, were unable to express the behaviour of concurrent systems without a shift of correspondence, based on the *scheduling* of executions [6]: from proof-nets as processes to proof-nets as *executions*, stressing the fact that the meaning of a proofs lies in its normal form, reached by cut elimination, while the meaning of a process is not one of its *multiple* irreducible forms, but how each form is reached: which channels communicate, and what is the form of the process at each step. The aim of our work is to try to carry out the same correspondence, but in the setting of *Ludics* [12] to overcome the limitations of cut elimination by using a logical system with **interaction** at its core.

Ludics has already been related to process calculi in works by C. Faggian and co.: from this connection the authors aim at gaining a way to represent replication (as in [2]) and non determinism inside Ludics – whose standard version is lacking both; however the correspondence between execution and interaction is not developed at all. As we understand now, our work is rather taking the opposite direction of [9], by effectively representing the *causal order* and *conflict relation* of event structures in Ludics using *behaviours*. This is achieved using a new construction, found in [10], called *directed behaviour*, that allows controlled interaction paths from carefully *pruned* designs. Finally we form a strong correspondence between execution in CCS and interaction in Ludics, without losing neither its non-determinism nor its non-confluence (i.e. multiple normal forms).

Outline. After background notions and notations for both Ludics (designs, behaviours and interaction) and MCCS (a simple fragment of CCS), we define in the third section our interpretation $[\![P]\!]$ of a process P in term of a Ludics' *behaviour* equipped with an assignment function. Section 4 summarizes the main results for

MCCS and finishes with their extension to replication-free CCS. At first we focus on the correspondence between process execution and Ludics interaction: $[\![P]\!]$ characterizes all executions on P. We then present the parallel composition as merging of interpretations, and an operational version of the interpretation corresponding to the process execution. We also give a characterization of deadlock-free processes, in the form of a sufficient condition to check on the interpretation for deadlock-freedom. In the last section we explain how to apply our technique to full CCS, that is an extension to replication using Terui's computational Ludics [20].

2 Background

Ludics is an abstraction of multiplicative-additive linear logic proofs in sequent calculus, under focused discipline (used for optimizing proof-search space by Andreoli [1]), thus cut-free and with a strict alternation between **positive** and **negative** rules. *Designs* the objects of Ludics, replace formulas with *addresses* $\xi, \zeta, \ldots$ (sequences of natural numbers); subformulas becomes *sub-addresses* $\xi.1, \xi.2.1, \ldots$. To give an intuition, proofs are rewritten in the following way:

Example 2.1 *A* focused *proof of the formula* $((A_1 \otimes A_2)\,\&\,B)\parr C$ *is:*

$$\dfrac{\dfrac{\vdash A_1, C, \Delta_1 \qquad \vdash A_2, \Delta_2}{\vdash A_1 \otimes A_2, C, \Delta} \qquad \vdash B, C, \Delta}{\vdash ((A_1 \otimes A_2)\,\&\,B)\parr C, \Delta}$$

where the connectives & *and* $\parr$ *are introduced at the same time. Going further, and considering only positive formulas, it becomes*

$$\dfrac{\dfrac{A_1^\perp \vdash C, \Delta_1 \qquad A_2^\perp \vdash \Delta_2}{\vdash A_1 \otimes A_2, C, \Delta} \qquad \vdash B, C, \Delta}{((A_1 \otimes A_2)^\perp \oplus B^\perp) \otimes C^\perp \vdash \Delta}$$

Finally in Ludics, formulas are forgotten, and we have

$$\dfrac{\dfrac{\xi.1.1 \vdash \xi.3, \Delta_1 \qquad \xi.1.2 \vdash \Delta_2}{\vdash \xi.1, \xi.3, \Delta} \qquad \vdash \xi.2, \xi.3, \Delta}{\xi \vdash \Delta}$$

where the Δ_i*s are sets of addresses.*

Formally, designs can be defined thus:

Definition 2.2 *A* design $\mathcal{D}$ *is a tree of* abstract sequents *built and labeled by rules called actions, such that each branch ends with a positive action. The rules are the following:*

- Daimon *(*Dai^+_Δ *:)* $\dfrac{}{\vdash \Delta}$ ✠
  where Δ *is a finite set of addresses. The Daimon is a* positive *action.*

- Positive action: $\dfrac{\dots \quad \xi.i \vdash \Delta_i \quad \dots}{\vdash \Delta, \xi}\ (+, \xi, I)$
  where $i \in I$, *with* $I \subset \mathbb{N}$. I *is the* ramification *of the rule and* ξ *is its* focus. $\Delta_i s$ *are* disjoint *and* included *in* Δ.
- Negative action: $\dfrac{\dots \quad \vdash \xi.I, \Delta_I \quad \dots}{\xi \vdash \Delta}\ (-, \xi, \mathcal{N})$
  where $\mathcal{N}$ *is the ramification of the rule, and* ξ *its focus.* $\mathcal{N} \subset \mathscr{P}_{fin}(\mathbb{N})$, $\xi.I = \xi.1, \dots, \xi.n$, *with* $1, \dots, n \in I$ *and* $I \in \mathcal{N}$; *the* $\Delta_I s$ *are* included *in* Δ.

The ramification *denotes the number of branches generated by the rule; i.e. the premises containing the sub-addresses* $\xi.i$, *with* $i \in I$, *or* $\zeta.i_1, \dots, \zeta.i_n$ *with* $i_1, \dots, i_n \in I$ *and* $I \in \mathcal{N}$. *Any rule with a sub-address* $\xi.i$ *as focus is* justified *by the rule introducing* ξ.

Positive and negative actions are strictly alternated *and* Dai^+ *can only be the* last *action of a design.*

Interaction is the equivalent in Ludics of cut elimination, and define its dynamic; the way a design interact is explicited by its syntax, closing the gap with semantics. We use the presentation and definitions found in [17], essentially the same as the seminal article [12]. The following example gives an idea of two orthogonal designs and interaction between them:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\xi.1.1.1 \vdash}}{\vdash \xi.1.1}\ (+, \xi.1.1, \{1\})^4}{\xi.1 \vdash}\ (-, \xi.1, \{\{1\}\})^3 \qquad \dfrac{\dfrac{\dfrac{}{\xi.2.1.1 \vdash}}{\vdash \xi.2.1}\ (+, \xi.2.1, \{1\})^8}{\xi.2 \vdash}\ (-, \xi.2, \{\{1\}\})^7}{\vdash \xi}\ (+, \xi, \{1,2\})^0$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \xi.2.1.1, \xi.1.1.1}\ \maltese^{10}}{\xi.2.1 \vdash \xi.1.1.1}\ (-, \xi.2.1, \{\{1\}\})^9}{\vdash \xi.1.1.1, \xi.2}\ (+, \xi.2, \{1\})^6}{\xi.1.1 \vdash \xi.2}\ (-, \xi.1.1, \{\{1\}\})^5}{\vdash \xi.1, \xi.2}\ (+, \xi.1, \{1\})^2 \qquad \dfrac{}{\xi \vdash}\ (-, \xi, \{\{1,2\}\})^1$$

Here the numbers denote the n-th interaction step. Interaction starts from the cut on the bases (i.e. conclusions $\vdash \xi$ and $\xi \vdash$), and checks the premises of the $+$ rule: if the ramification $\{1,2\}$ finds a match in the corresponding negative rule, interaction continues, and ends *successfully* if it reaches a **daimon** $\maltese$: it fulfills the role of axioms, stopping proof-search. In case of successful interaction between two designs $\mathcal{D}$ and $\mathcal{C}$, we say that they are **orthogonals**, denoted $\mathcal{D} \perp \mathcal{C}$ or $\mathcal{C} \in \mathcal{D}^\perp$ (and vice-versa). We will interpret processes as *behaviours*; formally we have

Definition 2.3 *A* ***behaviour*** *is a set of designs* $\mathscr{B} = \{\mathcal{D}_1, \dots, \mathcal{D}_n\}$ *of same base* closed by bi-orthogonality, *i.e. such that* $\mathscr{B}^{\perp\perp} = \mathscr{B}$.

MCCS, a fragment of Milner's CCS. At first we restrict our setting to the *multiplicative* fragment of CCS, as presented in [4], in short called MCCS, to keep it as simple as possible. In section 4.5 the same tools will be used to represent non-deterministic choice and *name* hiding (aka *restriction*). Replication is addressed in section 5 and treated in a reformulated version of Ludics by K. Terui, called c-ludics [20]. MCCS terms are described by the following syntax:

$$P, Q := 1 \mid a^l.P \mid \overline{a}^m.P \mid (P \mid Q)$$

$a, b, c, \ldots$ denotes *channel names* occurrences, taken from a countable set $\mathcal{A}$ of names, and are labeled with *locations* $l, m, o, \ldots$, taken from a countable set Loc of locations (Loc_P denotes the locations of a process P). They are denoted a^l, b^m, c^o, etc. $a^l.P$ is the *positive* action prefix, $\overline{a}^m.P$ the *negative* one, and $P \mid Q$ the *parallel composition* of two processes. We use 1 to denote the *empty process*, instead of the traditional 0, since it is not only the neutral element of the non-deterministic choice (+), but also of the parallel composition ($P \mid 1 \equiv P$), which shows a *multiplicative* behaviour in our interpretation (as we will see, there is also a connection with the linear logic unit 1). A partial order on locations $<_P$ is induced by the *prefix order* of the occurrences of channel names they label: if $P = a^l.b^m \mid Q$, then $l <_P m$. A *synchronization*, denoted $i, j, u, v, \ldots$, is a pair $(a^l, \overline{a}^m)$ of dual channel-name occurrences, which we can synchronize to perform *execution*, by the local rule

$$a^l.P \mid \overline{a}^m.Q \rightarrow_{(l,m)} P \mid Q.$$

The set of synchronizations of P is denoted $\mathcal{S}_P$. If two synchronizations u, v have a location in common, then $(u, v) \in \mathcal{X}_P$, the set of *xor* conditions of P, a conflict-like relation. With $xor(u) = \{v_1, \ldots, v_n \mid (u, v_i) \in \mathcal{X}_P, 1 \leq i \leq n\}$ we denote the set of synchronizations in conflict with u, i.e. the ones that cannot be in the same execution sequence with u. The partial order on locations $<_P$ is naturally extended to synchronizations and denoted $\preccurlyeq_{\mathcal{S}_P}$: $u \preccurlyeq_{\mathcal{S}_P} v$ if there is a location $l \in u$ and a location $m \in v$ such that $l <_P m$. Equality comes from the fact that a location of u could be smaller or greater than a location of v and vice-versa, forming a cycle $u \preccurlyeq v \preccurlyeq u$ in the order.

3 The interpretation

We present here the interpretation of the MCCS fragment. Firstly, the elements of the process relevant for execution must be represented: via an assignment of locations, cuts, and *xor* conditions to addresses, we can code them into designs. We interpret each element of Loc_P, $\mathcal{S}_P$ and $\mathcal{X}_P$ into *negative* designs, then put them together as premises (sub-designs) of a single positive design, the *base design*, denoted $\mathcal{D}_P$, that is a preliminary and naive interpretation of a process P.

Secondly, also the order and conflict relations between these elements need to be coded, and respected during interaction: a new operation on designs is used to restrict interaction paths accordingly to the said relations, and closure properties are obtained by bi-orthogonality, generating a behaviour. The partial orders $<_P$ and $\preccurlyeq_{\mathcal{S}_P}$ and the conflict relation $\mathcal{X}_P$ are represented using particular *directed* (or *non commutative*) modifications of $\mathcal{D}_P$, called *restriction designs*, denoted $\mathcal{R}(P)$. These modifications exploit a technique found in [10], the *pruning* of a branch of a design, which affects interaction in the context of a behaviour. The role of $\mathcal{R}(P)$ is restricting the possible interactions on $\mathcal{D}_P$ by forcing them to respect the prefix order and conflict relation, once we put the designs together as the set of generators of a *behaviour*. Using the designs of the previous example, we try to give an intuition

of the idea behind the pruning.

Example 3.1 $\mathcal{D}:$
$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\xi.1.1.1\vdash}}{\vdash \xi.1.1}\,(+,\xi.1.1,\{1\})}{\xi.1\vdash}\,(-,\xi.1,\{\{1\}\})\qquad \dfrac{\dfrac{\overline{\xi.2.1.1\vdash}}{\vdash \xi.2.1}\,(+,\xi.2.1,\{1\})}{\xi.2\vdash}\,(-,\xi.2,\{\{1\}\})}{\vdash \xi}\,(+,\xi,\{1,2\})$$

$\mathcal{C}:$
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\vdash \xi.2.1.1,\xi.1.1.1}}{\xi.2.1\vdash \xi.1.1.1}\,(-,\xi.2.1,\{\{1\}\})}{\vdash \xi.1.1.1,\xi.2}\,(+,\xi.2,\{1\})}{\xi.1.1\vdash \xi.2}\,(-,\xi.1.1,\{\{1\}\})}{\vdash \xi.1,\xi.2}\,(+,\xi.1,\{1\})}{\xi\vdash}\,(-,\xi,\{\{1,2\}\})$$

$\mathcal{E}:$
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\vdash \xi.1.1.1,\xi.2.1.1}}{\xi.1.1\vdash \xi.2.1.1}\,(-,\xi.1.1,\{\{1\}\})}{\vdash \xi.2.1.1,\xi.2}\,(+,\xi.1,\{1\})}{\xi.2.1\vdash \xi.2}\,(-,\xi.2.1,\{\{1\}\})}{\vdash \xi.2,\xi.1}\,(+,\xi.2,\{1\})}{\xi\vdash}\,(-,\xi,\{\{1,2\}\})$$

Both $\mathcal{C}$ and $\mathcal{E}$ are orthogonal to $\mathcal{D}$. However

$$\mathcal{D}^*:\ \dfrac{\dfrac{}{\xi.1\vdash}\ p\qquad \dfrac{\overline{\vdash \xi.2.1}}{\xi.2\vdash}\,(-,\xi.2,\{\{1\}\})}{\vdash \xi}\,(+,\xi,\{1,2\})$$

where p denotes a pruning *on the branch starting with $\xi.1\vdash$, is orthogonal only to $\mathcal{E}$: interaction cannot continue on $\xi.1$, since it is not introduced by a rule anymore, but can only pass through $\xi.2\vdash$, stopping at the ✠ above $\xi.2.1$. In conclusion, $\mathcal{E}\in\{\mathcal{D},\mathcal{D}^*\}^\perp$, since it visits the $\xi.2$ branch first, while $\mathcal{C}\notin\{\mathcal{D},\mathcal{D}^*\}^\perp$; in this way we have forced interaction to respect the order $\xi.2<\xi.1$.*

Formally, let $[\ ]_P$ be an assignment function from $Loc_P\cup\mathcal{S}_P\cup\mathcal{X}_P$ to addresses: for $x\in Loc_P\cup\mathcal{S}_P$, we build the negative design $G[x]$ described by the device of Fig.1 where $[x]_P$ is the address assigned to the synchronization or location in question; for instance $[u]_P=\xi.1$ and $[l]_P=\xi.2$, with $u\in\mathcal{S}_P$ and $l\in Loc_P$. Each $(u,v)\in\mathcal{X}_P$, to fork the interaction path, is interpreted by $w[u,v]$, in Fig.1, where we note the action as & since it is a binary negative rule, and $xor^u\,\&\,xor^v$ is an address assigned to the clause (u,v). Then, the base design is:

$$\mathcal{D}_P=(\bigotimes_{x\in(\mathcal{S}_P\cup Loc_P),(u,v)\in\mathcal{X}_P}\{G[x],w[u,v]\})$$

where $\otimes$ stands for a sole **positive** rule $(+,\xi,I)$ with each element of the set as an

$$G[x]=\dfrac{\dfrac{[x]_P.1.1\vdash}{\vdash [x]_P.1}}{[x]_P\vdash}\qquad w[u,v]=\dfrac{\dfrac{xor^u.1\vdash}{\vdash xor^u}\qquad \dfrac{xor^v.1\vdash}{\vdash xor^v}}{xor^u\,\&\,xor^v\vdash}$$

$$\mathcal{R}(i)=\dfrac{..\quad \dfrac{\dfrac{\overline{\vdash [l].1}}{[l]\vdash}\,✠\qquad ..\quad \dfrac{}{[i]\vdash}\,\mathrm{p}\quad ..}{\vdash \xi}$$

Fig. 1. Device designs, and restriction design for $i\in\mathcal{S}_P$.

element of the ramification I (we assume that the assigned addresses have all the same prefix ξ). Each negative design is thus a different premise, each containing a sub-address of the focus. A *restriction* $\mathcal{R}(i)$ for a synchronization $i = (a^m, \overline{a}^o)$ such that, for instance, $l <_P o$, is an alteration of a copy of $\mathcal{D}_P$ obtained by forcing a ✠ on $G[l]$ and a pruning on $G[i]$. Such restriction $\mathcal{R}(i)$ is of the form described in Fig.1, where the dots $\cdots$ stands for all the other sub-designs of $\mathcal{D}_P$, which remain unaffected. Any interaction with a design orthogonal to both $\mathcal{R}(i)$ and $\mathcal{D}_P$ will be forced to visit $[l]\vdash$ before $[i]\vdash$. For $(u,v) \in \mathcal{X}_P$, instead, we need two restriction designs described in Fig.2; thus tying each member of the *xor* pair (u or v) to a different branch of the negative rule, which effectively fork the interaction path. To finish we need to put the base design and all restriction designs together: they form *the generators* of a behaviour, obtained by bi-orthogonality. Therefore, let

$$\mathscr{B}_P = \mathbb{B}_P^{\perp\perp} = (\{\mathcal{D}_P\} \cup \mathcal{R}(x) \cup \mathcal{R}((u,v)))^{\perp\perp}$$

for all $x \in (\mathcal{S}_P \cup Loc_P)$, $(u,v) \in \mathcal{X}_P$ (note that $\mathcal{R}(x/(u,v))$ is actually a *set* of restrictions, usually more than one design). Then, the **interpretation of P** is $[\![P]\!] = (\mathscr{B}_P, [\ ]_P)$, the pair formed by the behaviour $\mathscr{B}_P$ and an assignment function $[\ ]_P$ from $\mathcal{S}_P$, $\mathcal{X}_P$ and Loc_P to addresses.

4 Main results

In this section we present results for MCCS, and will extend the setting to replication-free CCS only later, for which all the results remain valid. Solutions for the extension to replication are presented in the next section, giving a correspondence for full CCS.

The results consist of the expected correspondence between process execution and interaction in Ludics (Theorem 4.3), and between parallel composition and the merging of interpretations – that is, the operation $\otimes$ on behaviours, the ludical correspective of the linear logic tensor – when composing independent processes (Theorem 4.5); then, an operation mimicking execution on behaviours, giving us an intended weak subject reduction property (Theorem 4.6), and a characterization of deadlock-free processes, based on the notion of *visitable paths* of a behaviour (Theorem 4.8).

$$\dfrac{\dfrac{\dfrac{}{\vdash xor^u} \ ✠ \ \dfrac{xor^v.1 \vdash}{\vdash xor^v}}{xor^u \,\&\, xor^v \vdash} \quad \dfrac{}{[u]\vdash}\,p}{\vdash \xi} \qquad \dfrac{\dfrac{\dfrac{xor^u.1 \vdash}{\vdash xor^u} \quad \dfrac{}{\vdash xor^v} \ ✠}{xor^u \,\&\, xor^v \vdash} \quad \dfrac{}{[v]\vdash}\,p}{\vdash \xi}$$

Fig. 2. Two restriction designs for $(u,v) \in \mathcal{X}_P$.

4.1 Correspondence between execution and interaction

In order to form a correspondence between the *dynamic* of a process P and its interpretation $[\![P]\!]$ we need to be able to extract from interaction the relevant information describing execution. The core notion is the one of *visited actions* inside an *interaction path*. The actions considered at each step by interaction are said *visited*, and an *interaction path* on a design is the sequence of visited actions. Our aim is to give a definition of *associated execution* which will make any orthogonal design describe an execution sequence, even if the empty one; at the same time the directed restrictions will ensure that *if* a design is orthogonal to $\mathscr{B}_P$ *then* its associated execution on the process P is *admissible*, i.e. it respects the partial order on synchronizations $<_{\mathcal{S}_P}$, and the *xor* conditions $\mathcal{X}_P$, thus being a possible execution sequence on P. The following notion is therefore well defined. Let $\mathcal{K}^{\mathcal{D}}_{\mathcal{C}}$ be the sequence of actions of $\mathcal{D}$ visited during interaction with $\mathcal{C}$, we have

Definition 4.1 *Let $\mathcal{C} \in \mathscr{B}_P^{\perp}$. The execution on P* associated *to $\mathcal{C}$ is $\to_{i_1} \cdots \to_{i_n}$, the execution sequence such that for all synchronization $i \in (i_1, \ldots, i_n)$,*

$$(-,[i],\{\{1\}\})(+,[i].1,\{1\}) = G[i] \in \mathcal{K}^{\mathcal{D}_P}_{\mathcal{C}},$$

ordered as they are visited by the interaction path.

4.1.1 An Example of associated execution

Example 4.2 *Let $\mathcal{D}_P$ =*

$$\dfrac{\cdots \quad \dfrac{\dfrac{\dfrac{\overline{\xi.1.1.1 \vdash}}{\vdash \xi.1.1.1}\,(+,\xi.1.1,\{1\})^4}{\xi.1 \vdash}\,(-,\xi.1,\{\{1\}\})^3 \quad \dfrac{\dfrac{\overline{\xi.2.1.1 \vdash}}{\vdash \xi.2.1.1}\,(+,\xi.2.1,\{1\})^8}{\xi.2 \vdash}\,(-,\xi.2,\{\{1\}\})^7 \quad \cdots}{\vdash \xi}\,(+,\xi,I)^0}{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \xi.2.1.1, \xi.1.1.1, \Delta}{\xi.2.1 \vdash \xi.1.1.1, \Delta}\,(-,\xi.2.1,\{\{1\}\})^9}{\vdash \xi.1.1.1, \xi.2, \Delta}\,(+,\xi.2,\{1\})^6}{\xi.1.1 \vdash \xi.2, \Delta}\,(-,\xi.1.1,\{\{1\}\})^5}{\vdash \xi.1, \xi.2, \Delta}\,(+,\xi.1,\{1\})^2}{\xi \vdash}\,(-,\xi,\{I\})^1} \; \text{✠}^{10}$$

and $\mathcal{C}$ =

Assume $\xi.1 = [i]$ and $\xi.2 = [j]$. Then $\mathcal{K}^{\mathcal{D}_P}_{\mathcal{C}}$ =

$(+,\xi,I)(-,[i],\{\{1\}\})(+,[i].1,\{1\})(-,[j],\{\{1\}\})(+,[j].1,\{1\})$.

Therefore the execution associated to $\mathcal{K}^{\mathcal{D}_P}_{\mathcal{C}}$ is $\to_{i,j}$. Furthermore i, j are minimal synchronizations *with respect to $\leqslant_{\mathcal{S}_P}$, since they are visited first in $\mathcal{D}_P$ (and therefore also have no xor conditions).*

By construction of the interpretation and definition of associated execution, we find the expected correspondence between *execution* on processes and *interaction* on behaviours:

Theorem 4.3 *Let P be a MCCS process, $[\![P]\!]$ characterizes all executions on P.*

With *characterizes* we mean that to each interaction between $\mathscr{B}_P$ and $\mathscr{B}_P^{\perp}$ is associated an execution, and each execution is associated to *at least* one interaction. It correspond to an *admissible* execution since the restriction designs force interaction to respect the partial order $<_P$ and conflict relation $\mathcal{X}_P$.

4.2 Parallel composition as merging of interpretations

To make the translation **modular** we need to represent the parallel composition on processes $P \mid Q$ by a composition of their respective behaviours $\mathscr{B}_P$ and $\mathscr{B}_Q$, via a logical operation on them. This operation, called *merging*, is based on $\otimes$, the Ludics equivalent of the linear logic tensor $\otimes$, which is the extension to behaviours of the more primitive composition $\odot$ on positive designs, also called *merging* (of designs). Informally, let $\mathcal{D}$ and $\mathcal{C}$ be designs of the same base $\vdash \xi$ and of respective first action $(+, \xi, I)$ and $(+, \xi, J)$, such that $I \cap J = \varnothing$. Then,

$$\mathcal{D} \odot \mathcal{C} = \{(+, \xi, I \cup J)\mathrm{c} \mid (+, \xi, I)\mathrm{c} \in \mathcal{D} \text{ or } (+, \xi, J)\mathrm{c} \in \mathcal{C}\}$$

where c denotes a branch (usually called a *chronicle*) of the design in question. If $I \cap J \neq \varnothing$ or either $\mathcal{D}$ or $\mathcal{C}$ are $\maltese$, then $\mathcal{D} \odot \mathcal{C} = \maltese$.

The full operation is defined thus:

Definition 4.4 *[17] Let $\mathscr{B}$ and $\mathscr{E}$ be* positive *behaviours, with disjoint ramifications of the first rule, and of same base. Then*

$$\mathscr{B} \bigotimes \mathscr{E} = \{\mathcal{D} \odot \mathcal{C} \mid \mathcal{D} \in \mathscr{B}, \mathcal{C} \in \mathscr{E}\}^{\perp\perp}$$

This operation, along with a composition of the assignments $[\ \ ]_P \cup [\ \ ]_Q$ and a few intermediate steps, let us compose the interpretations of two processes P and Q to achieve the interpretation of $P \mid Q$. In the trivial case where P and Q cannot communicate, it is a straight correspondence:

Lemma 4.1 *Given $\mathscr{B}_P$ and $\mathscr{B}_Q$ as behaviours with the same base, and disjoint ramifications of the first rule, if there is no communication between P and Q, then the corresponding interpretation is*

$$[\![P \mid Q]\!] = (\mathscr{B}_P \bigotimes \mathscr{B}_Q, [\ \ ]_P \cup [\ \ ]_Q).$$

The assumption poses no issues, since the addresses assigned by the function $[\ \ ]_P$ are completely arbitrary, and it only matters which element is assigned to which address. We may also assume a renaming of either $[\ \ ]_P$ or $[\ \ ]_Q$ to have them *disjoint*: this would let $\mathcal{D}_P$ and $\mathcal{D}_Q$ have the same base, but a *disjoint* ramification of the first action; i.e. no common sub-address, and thus no conflict in the assignments $[\ \ ]_P$ and $[\ \ ]_Q$. If there are possible communications between P and Q a few modifications are needed on the construction of the interpretation, to account for the new synchronizations generated in the parallel composition $P \mid Q$, and their *xor* conditions; but the logical core operation $\odot$ is kept intact on the base designs. The

new objects that need to appear in the merging can be captured by the sets

$$new\mathcal{S}_{P|Q} = \mathcal{S}_{P|Q} \setminus (\mathcal{S}_P \cup \mathcal{S}_Q) \quad \text{and} \quad new\mathcal{X}_{P|Q} = \mathcal{X}_{P|Q} \setminus (\mathcal{X}_P \cup \mathcal{X}_Q).$$

We can deduce the *new* synchronizations, *xor* conditions and their restriction designs by checking $[\ \ ]_P$ and $[\ \ ]_Q$, which carry the information about channel names, with no need to know the structure of the processes. What we need to add is a design $\mathcal{N}_{P|Q}$ which accounts for the new element generated in the parallel composition. Given two processes P and Q, let $(+, \xi, I)$ and $(+, \xi, J)$ be the first action of, respectively, $\mathcal{D}_P$ and $\mathcal{D}_Q$; then $\mathcal{N}_{P|Q}$ is the following design, for $N \cap I = N \cap J = \varnothing$

$$\frac{G[k_1]\ \cdots\ G[k_n]\ \ w[x_1, y_1]\ \cdots\ w[x_n, y_n]}{\vdash \xi}\ (+, \xi, N)$$

where we have $\{k_1, \ldots, k_n\} = new\mathcal{S}_{P|Q}$, and $\{(x_1, y_1), \ldots, (x_n, y_n)\} = new\mathcal{X}_{P|Q}$.

With $[\ \ ]_{\mathcal{N}}$ we denote the assignment of $new\mathcal{S}_{P|Q}$ and $new\mathcal{X}_{P|Q}$ to addresses introduced by the ramification N. We then consider $\mathcal{D}_P \odot \mathcal{D}_Q \odot \mathcal{N}_{P|Q}$, and build the restriction designs on this extended base design in the same way, to make interaction respect $<_{P|Q}$, which is the union of the two partial orders $<_P \cup <_Q$ (no new location is generated), and $\mathcal{X}_{P|Q}$. The only new restrictions will be the ones about elements of $new\mathcal{S}_{P|Q}$ and $new\mathcal{X}_{P|Q}$. The result of the operation takes the base design $\mathcal{D}_P \odot \mathcal{D}_Q \odot \mathcal{N}_{P|Q}$ together with the restrictions re-built on it, to generate a behaviour by bi-orthogonal closure; the result of this operation is denoted $(\mathbb{B}_P \Cap \mathbb{B}_Q)^{\perp\perp}$. Then, by taking the union of the assignments we get the *merging of interpretations*:

$$[\![P]\!] \Cap [\![Q]\!] = ((\mathbb{B}_P \Cap \mathbb{B}_Q)^{\perp\perp}, [\ \ ]_{P \odot Q} \cup [\ \ ]_{\mathcal{N}}).$$

The following result is a generalization of the previous lemma:

Theorem 4.5 *Let P, Q be MCCS processes. We have $[\![P]\!] \Cap [\![Q]\!] = [\![P \mid Q]\!]$.*

4.3 Subject reduction property

An explanation of the interpretation can be found by understanding process execution inside $[\![P]\!]$. This can be seen through a subject reduction property. We define the *reduction* on $[\![P]\!]$ for a given $u \in \mathcal{S}_P$ as an operation that matches execution $\rightarrow_u$ on P, denoted $[\![P]\!] \rightsquigarrow_u ([\![P]\!])_u$. Technically, we use an operation called *trimming*, that is a carefully defined *projection* on behaviours (originally defined in [12] and [17]), essentially a removal of a sub-ramification from the first action $(+, \xi, I)$ of $\mathcal{D}_P$ (operation that affects the whole behaviour, since it is built on $\mathcal{D}_P$). Informally, reduction is defined by erasing a sub-ramification, thus the entire subdesigns $G[x]$ or $w[x, y]$, corresponding to a designed synchronization $u = (a^l, \overline{a}^m)$, called the branches *associated* to u: $G[u]$, $G[l]$ and $G[m]$, $G[x]$ and $w[u, x]$ for $x \in xor(u)$, and $w[x, y]$ for $y \in xor(x)$. Let $K = \{i, \ldots, m\}$ be the corresponding sub-ramification, then the first action of $\mathcal{D}_P$ becomes $(+, \xi, I \setminus K)$.

As a consequence, all the $\mathcal{R}(P)$ will miss the same sub-designs, as well as the whole behaviour $\mathscr{B}_P$. There is a sort of inclusion of $(\llbracket P \rrbracket)_u$ in $\llbracket P \rrbracket$ from the point of view of the possible interactions and associated executions; that is, for each interaction path $\mathcal{K}_{\mathcal{C}'}$ on $(\llbracket P \rrbracket)_u$, there is an interaction path $\mathcal{K}_{\mathcal{C}}$ on $\llbracket P \rrbracket$ such that the execution associated to $\mathcal{K}_{\mathcal{C}'}$ is either the same, or a *sub-execution*, of the one associated to $\mathcal{K}_{\mathcal{C}}$. This second case holds if $\mathcal{K}_{\mathcal{C}'}$ visits $G[v]$ for a synchronization v such that $u \preccurlyeq_{\mathcal{S}_P} v$ – since $G[u]$ has been erased in $(\llbracket P \rrbracket)_u$, then execution on $G[v]$ is directly possible, while on $\llbracket P \rrbracket$, $G[u]$ must be visited first. The first case holds, in general, since $(\mathcal{D}_P)_u$ is strictly smaller than $\mathcal{D}_P$, thus any interaction on $(\mathscr{B}_P)_u$ is also possible on $\mathscr{B}_P$. Note that the restrictions corresponding to the elements *associated* to u automatically disappear in the behaviour $(\mathscr{B}_P)_u$: since the branches of these elements are erased, there are no more ✣ and prunings in the restrictions in question, making them exactly *equal* to $(\mathcal{D}_P)_u$.

The reduction on the interpretation is the operational side of our interpretation:

Theorem 4.6 *Let P be a MCCS process interpreted by $\llbracket P \rrbracket$. Given a synchronization $u \in \mathcal{S}_P$ such that $P \to_u P'$, we have $\llbracket P' \rrbracket = (\llbracket P \rrbracket)_u$, i.e.*

$$\begin{array}{ccc} P & \to_u & P' \\ {\scriptstyle \llbracket - \rrbracket}\downarrow & & \downarrow{\scriptstyle \llbracket - \rrbracket} \\ \llbracket P \rrbracket & \rightsquigarrow_u & (\llbracket P \rrbracket)_u \end{array}$$

The interpretation is thus not preserved during execution, but this must be the case if we want to fully represent the *dynamic* of a process. Indeed we preserve the *non-confluence* of execution: $\llbracket P \rrbracket$ keeps all the possible executions to normal forms of P. As a consequence, for some maximal execution sequences, we have:

Corollary 4.7 *Let $\mathbb{O}ne = \dfrac{}{\vdash \xi}^{(+,\xi,\varnothing)}$ (the design generating the behaviour that corresponds to the linear logic multiplicative unit 1), and let P be a MCCS process:*

$$\textit{if } P \to^* 1 \textit{ then } \llbracket P \rrbracket \rightsquigarrow^* \{\mathbb{O}ne\}^{\perp\perp} \textit{ for the same synchronization sequence.}$$

4.4 Deadlocks

By restricting a behaviour interpreting a process P to the set of its **visitable paths**, i.e. the branches of $\mathscr{B}_P$ which are actually visited by some interaction with $\mathscr{B}_P^{\perp}$, we obtain what is called the *incarnation* of a behaviour, which can give us some important information about the process. The result can be seen as a projection on $\mathscr{B}_P$ that erases the sub-ramification – and thus the sub-designs – *never* visited during interaction. We can restrict the definition to the *incarnation* of $\mathcal{D}_P$, denoted $|\mathcal{D}_P|$, i.e. the branches of $\mathcal{D}_P$ visited by some interaction, with the following consequences:

Theorem 4.8 (i) *If $|\mathcal{D}_P| = \mathcal{D}_P$, then P is* deadlock free.

(ii) *If $|\mathcal{D}_P| \neq \mathcal{D}_P$, then the* non visitable *part of $\mathcal{D}_P$ is never accessed by interaction, and its process counterpart represent the common part of all normal forms,*

where no synchronizations can occur (for any execution path); moreover, P is not reducible to 1.

A process P is *deadlocked* if P is in *normal form*, $P \neq 1$, and there is also a *cycle* in the partial order $\leqslant_{\mathcal{S}_P}$ on synchronizations; other definitions focus on the lack of communication on certain chosen occurrences of channel names (as for instance in [13]). In case (i), P is *deadlock free* since any part of P is potentially synchronizable (is visitable in $[\![P]\!]$). We don't know if $P \to^* 1$, but for each channel there is at least one execution path where it is synchronized with a dual, i.e. any part of P can be accessed by some execution. Thus, there are no *cycles* in the order $\leqslant_{\mathcal{S}_P}$, and we can potentially communicate with any channel of P (i.e for all channels there is a synchronizing execution sequence).

In case (ii), for the correspondence between interaction and execution, if a part of $\mathcal{D}_P$ is never visited, then there are some channels of the process that cannot be synchronized (either there is no dual, a deadlock, or the execution is blocked for some other reason) and, since each occurrence of a channel name is represented in $[\![P]\!]$, we also know which these channels are. Notice that $|\mathcal{D}_P|$ is the only relevant design to check, since the restriction designs are modifications of $\mathcal{D}_P$ only needed to restrain the possible interactions, and interaction-order. Thus, if an interaction visits any branch of a restriction $\mathcal{R}(\cdots)$, then it visits the same branch on $\mathcal{D}_P$.

For the same correspondence, we can interpret deadlocked processes in behaviours with no issues: the deadlocked part will just be *non visitable* in $[\![P]\!]$. For example:

$$P = a^1.\overline{b}^2.Q \mid b^3.\overline{c}^4.R \mid c^5.\overline{a}^6.S$$

is deadlocked, with a cycle $u_1 = (a^1, \overline{a}^6) \leqslant_{\mathcal{S}_P} u_2 = (\overline{b}^2, b^3) \leqslant_{\mathcal{S}_P} u_3 = (\overline{c}^4, c^5) \leqslant_{\mathcal{S}_P} u_1$.

On the side of $[\![P]\!]$, we have restrictions which will prevent us to interact with any u_i, since each $G[u_i]$ $(1 \leq i \leq 3)$ have as requisite a $G[l]$ for a location l of another synchronization in the cycle, which is not accessible as well for the same reason; therefore, there will be no orthogonal design interacting with any $G[u_i]$.

4.5 Extension to replication-free CCS: Non-deterministic Choice and Hiding

We present here the replication-free fragment of CCS, and the extension of the previous results. The non-deterministic choice + (also called *sum*) is a mutual exclusion between its two members; it waits for an *external* choice, i.e. a context in parallel composition, which selects one process to use by synchronizing with the channels of one of the two members, dropping the other for that execution path. Execution is therefore generalized in the following way:

$$P = a^l.P' + b^m.Q' \mid \overline{a}^n.P'' + \overline{b}^o.Q'' \to_{(l,n)} P' \mid P''$$

From the point of view of interaction the interpretation is extended with a *xor* condition in $\mathcal{X}_P$ on the synchronizations on a and b, i.e $u = (a^l, \overline{a}^n)$ and $v = (b^m, \overline{b}^o)$. Indeed, once we have performed execution on one, the other is excluded from the same execution path. On the other hand, both are possible until a choice is made,

and choosing one of the two synchronizations u and v, by transitivity of the partial order $<_P$ on locations, necessarily exclude from any further execution all the internal synchronizations of P' (if v is chosen) or P'' (if u is chosen) – as in event structures, where the conflict relation is hereditary w.r.t. causal implication. This is effectively described by the *xor* relation already present in the MCCS interpretation, that can mimic the non-deterministic choice, by extending it to members of a sum +.

The hiding operator extends the term syntax with $\nu a(P)$, also known as *restriction* $P\backslash a$ in early texts. It declares that a channel name is bound and *private* inside its scope, then hidden, i.e. that cannot communicate with channels outside its scope. If $P = \nu a(a^l.R) \mid \overline{a}^m.Q$ then the pair $(a^l, \overline{a}^m)$ cannot synchronize, and hence the channels would not be able to communicate. The execution rule is then considered under the scope of ν operators, up to common structural equivalence, which let push the hiding inside a process, until it reaches its maximal/minimal scope:

$$\nu a(P \mid Q) \equiv \ \nu a(P) \mid Q, \text{ if } a \notin fv(Q),$$
$$\nu a(P) \mid \nu b(Q) \equiv \nu a(\nu b(P \mid Q)) \equiv \nu b(\nu a(P \mid Q)), \text{ if } a \notin fv(Q) \text{ and } b \notin fv(P).$$

where fv notes the un-bound channels of a process. For a simple interpretation of hiding, we already have all the needed material: we restrict our definition of synchronizations so that only some pairs of dual channels are considered. We can denote with a^l_ν such a bound name; then we exclude from $\mathcal{S}_P$ any pair $(a^l, \overline{a}^m)$ such that *only one* channel is tagged, i.e. either is a^l_ν or $\overline{a}^m_\nu$. If both are tagged then it is still a synchronization. This implies that inside $\mathcal{D}_P$ and $\mathcal{R}(P)$ there is no trace at all of the hiding that can occur in the process P, we can only check its presence from the static assignment $[\ ]_P$, which only tells us singularly which channels occurrences are hidden. The result is that we forbid some interaction paths by *not interpreting*, instead of resorting to more restrictions.

5 Replication: a reformulation in computational Ludics

To handle replication one can follow the ideas of [21] to type event structures, by considering a restricted version of the π-calculus [1], a linear typed version of Sangiorgi's πI-calculus [19]. Such calculus has the same expressive power as the version with free name passing, and linearity only breaks for replicated outputs, but determinism is preserved by the uniqueness of inputs. This allow us to keep both $\preccurlyeq_{\mathcal{S}_P}$ and locations for synchronizations, even with replication. By this way the presented tools can be applied.

Another possibility to represent replication comes from an already existing version of Ludics dedicated for this. K. Terui formulated a complementary syntax for Ludics called *computational Ludics* [20] (aka c-ludics), closer to higher order π-calculus, to achieve practical advantages with the general goal of developing an

[1] Notice that [21] introduces such calculus to bypass the main difficulty to extend CCS semantics to the π-calculus: to switch from dynamic α-conversion that allows to represent the dynamic creation of a name, to a static one at typing time.

interactive theory of computability and *complexity* based on Ludics. The feature that seems most interesting to us is the possibility to represent infinite designs by a *finite generator*, allowing *recursive definitions*. Design generators let us easily extend the interpretation to the full calculus with replication.

Terui's syntax is based on a *signature* $\mathscr{A} = (A, ar)$, where A is a set of *names*, and $ar : A \to \mathbb{N}$ is a function giving an arity to each name. A denumerable set of variables V is needed, denoted $x, y, z, \ldots$. A *positive action* is either $\maltese$, Ω (noting the divergence), or $\overline{a}$, with $a \in A$; a *negative* action is $x \in V$, or $a(x_1, \ldots, x_n)$, with $a \in A$ and $ar(a) = n$. $x_1, \ldots, x_n$ are distinct variables, and $\overrightarrow{x}_a$ denotes a vector of variables of the arity of a. Informally, a design $\mathcal{D}$ is co-inductively defined by

$$P ::= \maltese \mid \Omega \mid (N_0 \mid \overline{a}\langle N_1, \ldots, N_n\rangle),$$
$$N ::= x \mid \sum a(\overrightarrow{x}_a).P_a.$$

P denotes the *positive* actions, N the negative actions. A name denotes both the polarity and cardinality of the ramification of a rule, and, in the negative rule, the variables stand for each sub-address of the ramification. If N_0 is not a variable x in a positive design, then it becomes a *cut*.

A reformulation of the designs used in the translation is possible, since it holds the following:

Remark 5.1 ***Standard*** *c-designs*[2] *(i.e. $\neq \Omega$, linear, cut-free and identity-free) correspond to the original designs.*

Consequently, it is easy to show that using the pruning to build restriction designs can naturally be applied to c-designs. Assuming an assignment from $Loc_P, \mathcal{S}_P, \mathcal{X}_P$ to names, we have the following correspondence:

- $G[u] = [u](x_u).(x_u \mid \overline{[u.1]}\langle 0\rangle)$.
- $w[u, v] = [xor^u](x_u).x_u \mid \overline{[xor^u.1]}\langle 0\rangle + [xor^v](x_v).x_v \mid \overline{[xor^v.1]}\langle 0\rangle$.
- $\mathcal{D}_P = x_0 \mid \overline{a}\langle G[x], \ldots, w[x, y], \ldots\rangle$, with x varying on $\mathcal{S}_P \cup Loc_P$, and (x, y) on $\mathcal{X}_P$;

where $[\ ]$ denotes the assignment function. Interaction is called *reduction*, and is defined in λ-calculus style on positive c-designs with a cut, by:

$$(\sum a(\overrightarrow{x_a}.P_a) \mid \overline{a}\langle\overrightarrow{N}\rangle) \to P_a[\overrightarrow{N}/\overrightarrow{x_a}]$$

where $\overrightarrow{N}$ is of lenght $ar(a)$. The reduction relation select the $a(\overrightarrow{x_a})$ that matches with $\overline{a}\langle\overrightarrow{N}\rangle$, thus assuring us that they have the same arity. Then, in the corresponding P_a, each variable is substituted by a negative design inside the scope of $\overline{a}$; reduction can then continue on $P_a[\overrightarrow{N}/\overrightarrow{x_a}]$, until a normal form is reached (a variable x or $\maltese$), or it diverges (Ω). Using the reduction relation, we can rewrite example 3.1 using prunings and $\maltese$ easily:

- $\mathcal{D} = x_0 \mid \overline{a}_0\langle a_1(x_1).x_1 \mid \overline{a_{11}}\langle 0\rangle,\ a_2(x_2).x_2 \mid \overline{a_{21}}\langle 0\rangle\rangle$
- $\mathcal{C} = a_0(x_1, x_2).x_1 \mid \overline{a}_1\langle a_{11}(x_{11}).x_2 \mid \overline{a}_2\langle a_{21}(x_{21}).\maltese\rangle\rangle$.

[2] See [20], remark 2.2.

- $\mathcal{E} = a_0(x_1, x_2).x_2 \mid \overline{a}_2\langle a_{21}(x_{21}).x_1 \mid \overline{a}_1\langle a_{11}(x_{11}).\maltese\rangle\rangle$
- $\mathcal{D}^* = x_0 \mid \overline{a}_0\langle 0^p, a_2(x_2).\maltese\rangle$.

where ^{p} denotes a pruning, and 0 the empty negative action: it denotes that the + action has a premise, but there is no further action in the branch. Both reductions $\mathcal{C} \mid \mathcal{D}$ and $\mathcal{E} \mid \mathcal{D}$ reach $\maltese$ after five reduction steps. Instead, only $\mathcal{E}$ is orthogonal to $\mathcal{D}^*$. To perform reduction, we must substitute x_0 in $\mathcal{D}^*$ with $\mathcal{E}$, obtaining

$$\mathcal{E} \mid \mathcal{D}^* = (a_0(x_1, x_2).x_2 \mid \overline{a}_2\langle a_{21}(x_{21}).x_1 \mid \overline{a}_1\langle a_{11}(x_{11}).\maltese\rangle\rangle) \mid \overline{a}_0\langle 0^p, a_2(x_2).\maltese\rangle$$

Reduction reach $\maltese$ in only 2 steps:

(i) $(a_2(x_2).\maltese) \mid (\overline{a}_2\langle a_{21}(x_{21}).0^p) \mid \overline{a}_1\langle a_{11}(x_{11}).\maltese\rangle\rangle$.

(ii) $\maltese$.

Instead, the reduction

$$\mathcal{C} \mid \mathcal{D}^* = (a_0(x_1, x_2).x_1 \mid \overline{a}_1\langle a_{11}(x_{11}).x_2 \mid \overline{a}_2\langle a_{21}(x_{21}).\maltese\rangle\rangle) | \overline{a}_0\langle 0^p, a_2(x_2).\maltese\rangle$$

at the second step diverges:

(i) $(0^p) \mid (\overline{a}_1\langle a_{11}(x_{11}).\maltese) \mid a_2(x_2).\maltese \mid \overline{a}_2\langle a_{21}(x_{21}).\rangle\rangle$.

(ii) Ω; since 0^p has no P_0 and variables to perform the substitution on.

Therefore, the reformulation in *c*-ludics does not affect restriction designs, or the behaviour $\mathscr{B}_P$, and the correspondence between execution and interaction still holds. About the merging of interpretations, the operation $\odot$ is simply an extension of the arity of a positive rule – or a substitution with a name of the needed arity – by putting together in the scope of this action all the negative designs that we have. Thus from $\mathcal{D} = x_0 \mid \overline{a_1}\langle N_1, \ldots, N_k\rangle$ and $\mathcal{C} = x_0 \mid \overline{a_2}\langle N_{k+1}, \ldots, N_{k+n}\rangle$ we obtain $\mathcal{D} \odot \mathcal{C} = x_0 \mid \overline{a_3}\langle N_1, \ldots, N_k, N_{k+1}, \ldots, N_{k+n}\rangle$.

Trimming, instead, require us to erase the sub-ramification associated to a certain synchronization from $\mathcal{D}_P$, thus reducing the arity of the first action $\overline{a}$, and removing the corresponding negative designs (the dual operation of $\odot$). A substitution to a name of the right arity might be required, but the operation itself poses no issues. When reducing $\mathcal{D}_P$ to $\mathbb{O}ne$, the form we obtain is $\mathbb{O}ne = x_0 \mid \overline{a}\langle\rangle = x_0 \mid \overline{a}$, a positive action with a 0-ary name.

The main issue with c-ludics is that some difficulties arises when defining the assignment function, since names require modifications every time we act on $\mathcal{D}_P$, and variables are not absolute values. Both cases complicate the read back from $[\![P]\!]$ to elements and relations of the process. This forces the assignment to be *deduced* from the structure of a *c*-design, and not be independent from it anymore, while also losing the $1-1$ correspondence with elements of the process.

6 Conclusion

The interpretation of CCS into Ludics tries to overcome the problems, and satisfy the goals, that motivated our work. Its main properties are:

- A logical characterization of the full dynamic of processes without imposing functional behaviour, resorting to multiple translations by partially determinizing execution via scheduling, or sacrificing the non-determinism or non-confluence itself (by imposing linearity and other constraints on the syntax).
- A partial *modularity* in the interpretation, which let us combine the interpreting structures, behaviours in our case, as we do with processes via parallel composition. We are able to represent the composition via a ludical operation on behaviours that, in the trivial case where there is no communication between two processes, exactly interpret the linear logic tensor $\otimes$. Otherwise, some more artificial and non-ludical steps are required, by working on the assignment functions, but the core operation $\odot$ is kept intact on the base designs.
- Insights on the dynamics of processes, as expliciting what parallel composition entails when two process communicate, what causes forks in an execution paths, and how the different reduced forms of a process are related through their interpretations.

As a consequence of these properties, we have that:

- subject reduction describe a particular inclusion between the interpretations of a process and of one of its reduced forms, with respect to their structures and possible interactions;
- along with execution, *deadlocks* are also characterized. Instead of being a property of the interpretation – as it is with typing systems, where if a process is typable, then it is deadlock free – the interpretation in Ludics is oblivious of their presence, as is interaction on behaviours. Still, we have a way to know if a process is deadlock-free, or if it can't be reduced to 1, via the *visitable paths* of a behaviour.

Finally, the reformulation in c-ludics [20] let us have access to non-linearity and recursive definition in the form of *finite designs generators*, and thus extend the interpretation to the full calculus by using the same techniques presented here.

Another point requiring further investigation is the evident strong connection with *event structures*, which seem naturally represented by directed behaviours, generated via restriction designs of a base one. Indeed, event structures are indirectly already represented, passing through processes, since they are a model of CCS-like calculi [22].

References

[1] Andreoli, J., *Logic programming with focusing proofs in linear logic*, J. Log. Comput. **2** (1992), pp. 297–347.
URL http://dx.doi.org/10.1093/logcom/2.3.297

[2] Basaldella, M. and C. Faggian, *Ludics with repetitions (exponentials, interactive types and*

completeness), Logical Methods in Computer Science **7** (2011).
URL https://doi.org/10.2168/LMCS-7(2:13)2011

[3] Beffara, E., *A concurrent model for linear logic*, Electr. Notes Theor. Comput. Sci. **155** (2006), pp. 147–168.
URL http://dx.doi.org/10.1016/j.entcs.2005.11.055

[4] Beffara, E., *A logical view on scheduling in concurrency*, in: *Proceedings Fifth International Workshop on Classical Logic and Computation, CL&C 2014, Vienna, Austria, July 13, 2014.*, 2014, pp. 78–92.
URL https://doi.org/10.4204/EPTCS.164.6

[5] Beffara, E. and F. Maurel, *Concurrent nets: A study of prefixing in process calculi*, Theor. Comput. Sci. **356** (2006), pp. 356–373.
URL http://dx.doi.org/10.1016/j.tcs.2006.02.009

[6] Beffara, E. and V. Mogbil, *Proofs as executions*, in: *Theoretical Computer Science - 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings*, 2012, pp. 280–294.
URL http://dx.doi.org/10.1007/978-3-642-33475-7_20

[7] Caires, L. and J. A. Pérez, *Linearity, control effects, and behavioral types*, in: *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, 2017, pp. 229–259.
URL https://doi.org/10.1007/978-3-662-54434-1_9

[8] Ehrhard, T. and O. Laurent, *Interpreting a finitary π-calculus in differential interaction nets*, Inf. Comput. **208** (2010), pp. 606–633.
URL http://dx.doi.org/10.1016/j.ic.2009.06.005

[9] Faggian, C. and M. Piccolo, *Partial orders, event structures and linear strategies*, in: *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings*, 2009, pp. 95–111.
URL http://dx.doi.org/10.1007/978-3-642-02273-9_9

[10] Fouqueré, C. and M. Quatrini, *Study of behaviours via visitable paths*, Submitted to journal (2016).
URL https://arxiv.org/abs/1403.3772v3

[11] Girard, J., *Linear logic*, Theor. Comput. Sci. **50** (1987), pp. 1–102.
URL http://dx.doi.org/10.1016/0304-3975(87)90045-4

[12] Girard, J., *Locus solum: From the rules of logic to the logic of rules*, Mathematical Structures in Computer Science **11** (2001), pp. 301–506.
URL http://dx.doi.org/10.1017/S096012950100336X

[13] Kobayashi, N., S. Saito and E. Sumii, *An implicitly-typed deadlock-free process calculus*, in: *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, 2000, pp. 489–503.
URL https://doi.org/10.1007/3-540-44618-4_35

[14] Milner, R., "A Calculus of Communicating Systems," Lecture Notes in Computer Science **92**, Springer, 1980.
URL https://doi.org/10.1007/3-540-10235-3

[15] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, I*, Inf. Comput. **100** (1992), pp. 1–40.
URL https://doi.org/10.1016/0890-5401(92)90008-4

[16] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, II*, Inf. Comput. **100** (1992), pp. 41–77.
URL https://doi.org/10.1016/0890-5401(92)90009-5

[17] Quatrini, M., "La Ludique : une théorie de l'interaction, de la logique mathématique au langage naturel," Habilitation à diriger des recherches, Université Aix-Marseille (2014).
URL https://hal.archives-ouvertes.fr/tel-01234909

[18] Rozenberg, G. and P. S. Thiagarajan, *Petri nets: Basic notions, structure, behaviour*, in: *Current Trends in Concurrency, Overviews and Tutorials*, 1986 pp. 585–668.
URL https://doi.org/10.1007/BFb0027048

[19] Sangiorgi, D., *π-calculus, internal mobility, and agent-passing calculi*, Theor. Comput. Sci. **167** (1996), pp. 235–274.
URL http://dx.doi.org/10.1016/0304-3975(96)00075-8

[20] Terui, K., *Computational ludics*, Theor. Comput. Sci. **412** (2011), pp. 2048–2071.
URL http://dx.doi.org/10.1016/j.tcs.2010.12.026

[21] Varacca, D. and N. Yoshida, *Typed event structures and the linear π-calculus*, Theor. Comput. Sci. **411** (2010), pp. 1949–1973.
URL http://dx.doi.org/10.1016/j.tcs.2010.01.024

[22] Winskel, G., *Event structure semantics for CCS and related languages*, in: *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12-16, 1982, Proceedings*, 1982, pp. 561–576.
URL https://doi.org/10.1007/BFb0012800

[23] Winskel, G., *Event structures*, in: *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, 1986, pp. 325–392.
URL https://doi.org/10.1007/3-540-17906-2_31

[24] Winskel, G. and M. Nielsen, **4**, Oxford Clarendon Press, 1995.