Characterizing co-NL by a group action

Clément Aubert, Thomas Seiller

Under consideration for publication in Math. Struct. in Comp. Science

Characterizing co-NL by a group action

CLÉMENT AUBERT† and THOMAS SEILLER‡ [1]

† *Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, (UMR 7030), F-93430, Villetaneuse, France*

aubert@lipn.fr

‡ *I.H.É.S., Le Bois-Marie, 35, Route de Chartres, 91440 Bures-sur-Yvette, France*

seiller@ihes.fr

Received 15 September 2012 ; Revised 15 November 2013

In a recent paper, Girard (2012) proposed to use his recent construction of a geometry of interaction in the hyperfinite factor (Girard 2011) in an innovative way to characterize complexity classes. We begin by giving a detailed explanation of both the choices and the motivations of Girard's definitions. We then provide a complete proof that the complexity class **co-NL** can be characterized using this new approach. We introduce the nondeterministic pointer machine as a technical tool, a concrete model to compute algorithms.

1. Introduction

Traditionally, the study of complexity relies on the definition of programs based on some abstract machines, such as Turing machines. In recent years, a new approach to complexity stemmed from the so-called proofs-as-program – or Curry–Howard – correspondence which allows to understand program execution as a cut-elimination procedure in logic. This correspondence naturally extends to quantitative approaches that made it possible to work on complexity with tools coming from logic. Due to its resource-awareness, linear logic (**LL**) is particularly suitable to treat computational questions, and many bridges have been built between complexity classes and this formalism. To name a few, elementary linear logic (**ELL**) (Danos & Joinet 2003), soft linear logic (Lafont 2004) and bounded linear logic (Dal Lago & Hofmann 2010) characterize complexity classes, but only deterministic, sequential and equal to **P** (polytime) or above. New directions have recently been explored to characterize other complexity classes: **SBAL** (Schöpp 2007) characterizes **L** (logarithmic space), boolean proof nets (Aubert 2011, Terui 2004), was the first success toward a characterization of parallel classes.

[1] This work was partly supported by the ANR-10-BLAN-0213 Logoi, the ANR-08-BLAN-0211-01 Complice and the GDR-IM's 'visiting PhD student' Program.

All those attempts belong to the field of implicit computational complexity (ICC). One of the main advantages of the ICC approach is that it does not refer to a particular model or an external measuring condition. We only have to consider language restrictions (for instance by limiting the primitive recursion) or to infer the complexity properties of a program, for instance with techniques like quasi-interpretations. Linear logic offers a particularly nice framework to study complexity questions since the decomposition of implication into a linear implication and a duplication modality allows some fine tuning of the rules that govern the latter. All the previously quoted attempts are *implicit* characterization of complexity classes, as those logical system rest on the limitation of the computational power of **LL**. Next to the restrictions of recursion and the rewriting system with quasi-interpretation, this approach exhibits several interesting results as there is no need to perform the computation to know the space or time needed.

The *geometry of interaction* program (Girard 1989*b*) was introduced by Girard a few years after the introduction of **LL**. In a first approximation, it aims at giving an interpretation of proofs –or programs– that accounts for the dynamics of cut-elimination, hence of computation. Since the introduction of this program Girard proposed several constructions[1] to fulfill this program (Girard 1989*a*, Girard 1989*b*, Girard 2011). Due to the fact that they are centered around the notion of computation, these constructions are particularly adapted to study computational complexity (Baillot & Pedicini 2001, Dal Lago 2005).

The approach studied in this paper, which was proposed recently by Girard, differs from the previous works on complexity. Indeed, though it uses the tools of Girard's geometry of interaction in the hyperfinite factor (Girard 2011), its relation to the latter is restricted to the representation of integers which is, in this particular setting, uniform[2]: each integer is represented as an operator N_n in the hyperfinite type II_1 factor $\mathfrak{R}$. By using an operator-theoretic construction —the crossed product— it is possible to internalize some isomorphisms acting on $\mathfrak{R}$. These internalized isomorphisms can be understood as sort of 'basic instructions' one can use to define a sort of abstract machine. Such an abstract machine is thus an operator constructed using these basic instructions: the operators in the algebra generated by the internalizations of the isomorphisms. One can then define the language accepted by such an operator ϕ: the set of natural numbers such that the product ϕN_n is nilpotent. We will only refer to **co-NL** and won't use the famous result that it is equal to **NL** (Immerman 1988, Szelepcsényi 1987), because it is more natural to think of our framework as capturing *complementary* of complexity classes.

In this paper, we present in detail a first result obtained from this approach: considering the group of finite permutations of the natural numbers, we can obtain a characterization of the complexity class **co-NL**. To ease the presentation and proofs of the result, we will introduce *non-deterministic pointer machines*, which are a new characterization of **co-NL** in terms of abstract machines.

[1] The interested reader can find a more unifying approach in the second author's 'Interaction Graphs' (Seiller 2012*a*).

[2] All (size of) inputs are represented as object in a unique space, whereas the naive GoI interpretation of integers as matrices (see Section 2) would yield matrices of varying sizes, hence not all elements of a single algebra.

Outline

We start (Section 2) by explaining in detail, with numerous examples, how the proofs representing binary integers are represented by graphs in the setting of Geometry of Interaction, graphs that can be then seen as matrices. Computation will then be represented by the computation of the iterated products of a matrix PN where N is a matrix representing an integer and P is a matrix representing the program. The nilpotency of this product will represent the fact that the integer represented by N is accepted by the program represented by P.

But the representation of an integer as a matrix is non-uniform: the size of the matrix depends on the integer considered. Since we want the representations of programs to be able to handle any size of input, we embed the matrices representing integers into the hyperfinite factor, whose definition is recalled in Section 3. Operators that represent programs are constructed from finite permutations, which can be internalized –represented as operators– using the crossed product construction. This allows them to perform some very simple operations on the input. Namely, they will be able to cope with several copies of the input and to scan them independently. However, since the representation of integers in the hyperfinite factor is not unique, one needs the notion of normative pair (Subsection 4.2) to guarantee that a program is insensitive to the chosen representation of the integer.

We next introduce, in Section 5, a notion of abstract machines – non-deterministic pointer machines (*NDPM*) – well suited to be represented by operators. This model is very close to multi-head finite automata, a classical characterization of logspace computation, but we begin this section by presenting its specificities. We then prove that *NDPMs* can recognize any set in **co-NL** by providing an example of a **co-NL**-complete problem solved by a *NDPM* and a mechanism of reduction between problems.

We then define (Section 6) an encoding of *NDPMs* as a certain kind of operators –named boolean operators, which proves that **co-NL** is contained in the set of languages accepted by such operators. To show the converse, we first show that checking the nilpotency of a product PN in the hyperfinite factor, where P is a boolean operator and N represents an integer, is equivalent to checking that a certain matrix is nilpotent (this rest on the quite technical Lemma 31). Finally, we can show that deciding if this matrix is nilpotent is in **co-NL**.

2. Binary Integers

In this paper, we will be working with binary integers. In this section, we will explain how it is possible to represent these integers by matrices. As it turns out, representation by matrices is not satisfactory, and it will be necessary to represent integers by operators acting on an infinite-dimensional (separable) Hilbert space, as it will be done in Section 4.

In intuitionistic logic, binary lists are typed with $\forall X\ (X \Rightarrow X) \Rightarrow ((X \Rightarrow X) \Rightarrow (X \Rightarrow X))$. In **ELL**, the type of binary lists is:

$$\forall X\ !(X \multimap X) \multimap (!(X \multimap X) \multimap !(X \multimap X))$$

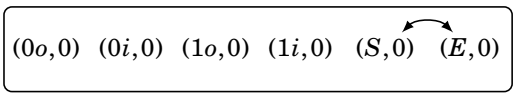
To a binary integer[3] corresponds a proof of the sequent $\vdash ?(X \otimes X^{\perp}), ?(X \otimes X^{\perp}), !(X \multimap X)$. One can easily read from a proof of this sequent the binary list it represents by looking at the occurences of contraction (and in some cases, weakening) rules. We develop below three examples: the empty list $\star$, the lists $\star 0$ and $\star 110$. In these examples, we labeled the variables in order to distinguish between the different occurrences of the variable X. This is necessary because we need to keep track of which formulas are principal in the contraction rule. This distinction, which by the way appears in geometry of interaction, is crucial since it may be the only difference between the proofs corresponding to two different binary lists. For instance, without this information, the proofs representing $\star 110$ and $\star 010$ would be exactly the same[4].

To each sequent calculus proof, we associate a graph which represents the axiom links in the sequent calculus proof. The vertices are arranged as a table where the different occurrences of the variables in the conclusion are represented on a horizontal scale, and a number of *slices* are represented on a vertical scale: the contraction is represented in geometry of interaction by a superimposition which is dealt with by introducing new copies of the occurrences using the notion of slices. In previous works (Seiller 2012*b*, Seiller 2012*a*), one of the authors showed how to obtain a combinatorial version of (a fragment) of Girard's geometry of interaction in the hyperfinite factor. Though the graphs shown here are more complex than the ones considered in these papers (in particular, the edges may go from one *slice* to another), they correspond exactly to the representation of binary lists in Girard's framework[5].

- The proof representing the empty list $\star$ uses the weakening rule twice:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash X(S)^{\perp}, X(E)}\,\text{ax}}{\vdash X(S) \multimap X(E)}\,\parr}{\vdash !(X(S) \multimap X(E))}\,!}{\vdash ?(X(0i) \otimes X(0o)^{\perp}), !(X(S) \multimap X(E))}\,?_w}{\vdash ?(X(0i) \otimes X(0o)^{\perp}), ?(X(1i) \otimes X(1o)^{\perp}), !(X(S) \multimap X(E))}\,?_w}{\vdash ?(X(0i)^{\perp} \multimap X(0o)^{\perp}), ?(X(1i) \otimes X(1o)^{\perp}), !(X(S) \multimap X(E))}\,\parr}{\vdash ?(X(0i)^{\perp} \multimap X(0o)^{\perp}), ?(X(1i)^{\perp} \multimap X(1o)^{\perp}), !(X(S) \multimap X(E))}\,\parr}{\vdash ?(X(0i)^{\perp} \multimap X(0o)^{\perp}), (!(X(1i) \multimap X(1o)) \multimap (!(X(S) \multimap X(E))))}\,\parr}{\vdash (!(X(0i) \multimap X(0o)) \multimap ((!(X(1i) \multimap X(1o)) \multimap (!(X(S) \multimap X(E)))))}\,\parr}{\vdash \forall X\, (!(X(0i) \multimap X(0o)) \multimap (!(X(1i) \multimap X(1o)) \multimap !(X(S) \multimap X(E))))}\,\forall$$

  We will use a double line in the following to ignore the bureaucracy of introducing all the $\parr$. The corresponding graph is:



[3] As binary lists trivially represent binary integers, we may focus on binary integers for free.

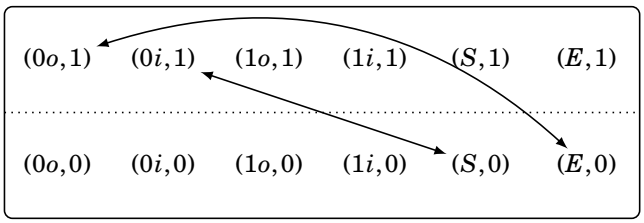
[4] Or more precisely, they would be indistinguishable. This is consequence of the fact that the contraction rule in sequent calculus does not distinguish which formulas it is contracting, even though it should in order to be correctly defined. For instance, there are more than one proof of $\vdash A, A$ obtained from $\vdash A, A, A$ by means of a contraction rule, but even though these are different, they are indistinguishable with the usual syntax of sequent calculus.

[5] For more details, one may consult Seiller's PhD Thesis (2012*c*).

- The proof representing the list $\star 0$ (resp. $\star 1$) uses a weakening to introduce $X(1i) \multimap X(1o)$ (resp. $X(0i) \multimap X(0o)$):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash X(S)^{\perp}, X(0i)}\,\mathrm{ax} \quad \dfrac{}{\vdash X(0o)^{\perp}, X(E)}\,\mathrm{ax}}{\vdash X(0i)\otimes X(0o)^{\perp}, X(S)^{\perp}, X(E)}\,\otimes}{\vdash X(0i)\otimes X(0o)^{\perp}, X(S)\multimap X(E)}\,\parr}{\vdash ?(X(0i)\otimes X(0o)^{\perp}), !(X(S)\multimap X(E))}\,!}{\vdash ?(X(0i)\otimes X(0o)^{\perp}), ?(X(1i)\otimes X(1o)^{\perp}), !(X(S)\multimap X(E))}\,?_w}{\vdash !(X(0i)\multimap X(0o)) \multimap (!(X(1i)\multimap X(1o)) \multimap !(X(S)\multimap X(E)))}}{\vdash \forall X\ !(X(0i)\multimap X(0o)) \multimap (!(X(1i)\multimap X(1o)) \multimap !(X(S)\multimap X(E)))}\,\forall$$

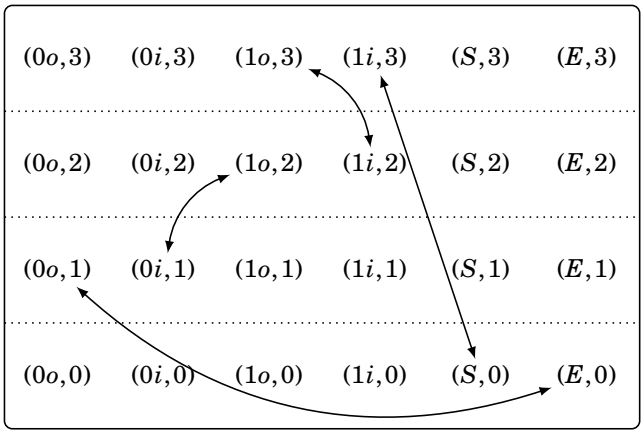
  The corresponding graph is:



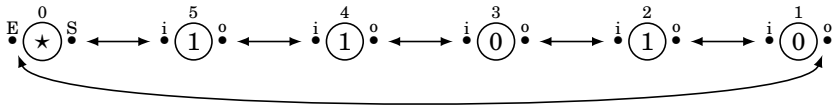
- The proof representing the list $\star 110$ contracts the occurrences $X(Ai)\otimes X(Ao)^{\perp}$ and $X(1i)\otimes X(1o)^{\perp}$, in bold below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash X(0i), X(E)^{\perp}}\,\mathrm{ax} \quad \dfrac{}{\vdash X(1i), X(0o)^{\perp}}\,\mathrm{ax}}{\vdash X(0i)\otimes X(0o)^{\perp}, X(1i), X(E)^{\perp}}\,\otimes \quad \dfrac{}{\vdash X(Ai), X(1o)^{\perp}}\,\mathrm{ax}}{\vdash X(0i)\otimes X(0o)^{\perp}, X(1i)\otimes X(1o)^{\perp}, X(Ai), X(E)^{\perp}}\,\otimes \quad \dfrac{}{\vdash X(S), X(Ao)^{\perp}}\,\mathrm{ax}}{\vdash X(0i)\otimes X(0o)^{\perp}, X(1i)\otimes X(1o)^{\perp}, X(Ai)\otimes X(Ao)^{\perp}, X(S), X(E)^{\perp}}\,\otimes}{\vdash X(0i)\otimes X(0o)^{\perp}, X(1i)\otimes X(1o)^{\perp}, X(Ai)\otimes X(Ao)^{\perp}, X(S)\multimap X(E)}\,\parr}{\vdash ?(X(0i)\otimes X(0o)^{\perp}), ?(\mathbf{X(1i)\otimes X(1o)^{\perp}}), ?(\mathbf{X(Ai)\otimes X(Ao)^{\perp}}), !(X(S)\multimap X(E))}\,!}{\vdash ?(X(0i)\otimes X(0o)^{\perp}), ?(\mathbf{X(1i)\otimes X(1o)^{\perp}}), !(X(S)\multimap X(E))}\,?_c}{\vdash !(X(0i)\multimap X(0o)) \multimap (!(X(1i)\multimap X(1o)) \multimap !(X(S)\multimap X(E)))}}{\vdash \forall X\ !(X(0i)\multimap X(0o)) \multimap (!(X(1i)\multimap X(1o)) \multimap !(X(S)\multimap X(E)))}\,\forall$$

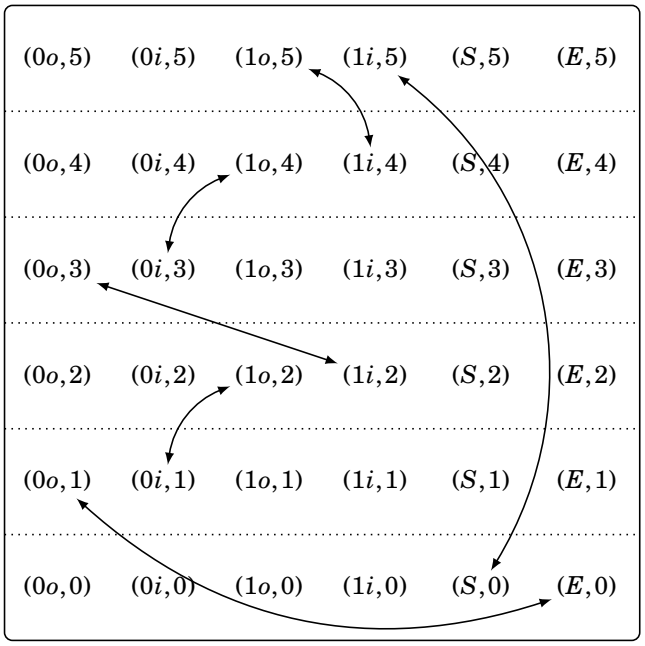
  The corresponding graph is:



The edges of the graph describe the scanning of the list. We illustrate this by explaining how to construct the graph corresponding to the list $\star 11010$. Indeed, the graph can be described directly from the list itself:



Each element of the list lives in a different *slice* —the integer shown above each element of the list. Moreover, each element is connected by its output vertex to its successor's input vertex (the successor of the last element is $\star$), and by its input vertex to its predecessor's output node. This gives the following graph, which is the representation of $\star 11010$:



Definition 1 (Matricial representation of a list)**.** Given a binary representation of an integer $n = \star a_1, \ldots, a_k$ of size $k \neq 0$[6] and its corresponding graph G_n, n is represented by M_n, a 6×6 block matrix of the following form:

$$M_n = \begin{array}{c} \overbrace{}^{0}\ \overbrace{}^{1}\ \overbrace{}^{*} \\ \begin{pmatrix} 0 & l_{00} & 0 & l_{10} & s_0 & 0 \\ l_{00}^* & 0 & l_{01}^* & 0 & 0 & e_0^* \\ 0 & l_{01} & 0 & l_{11} & s_1 & 0 \\ l_{10}^* & 0 & l_{11}^* & 0 & 0 & e_1^* \\ s_0^* & 0 & s_1^* & 0 & 0 & 0 \\ 0 & e_0 & 0 & e_1 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \}_0 \\ \}_1 \\ \}_* \end{array}$$

where coefficients are $(k+1) \times (k+1)$ matrices (the $(\cdot)^*$ denotes the conjugate-transpose) defined – for $u, v \in \{0,1\}$ – by:

[6] We will always assume in the following that the length of the binary integer representing the integer n under study is denoted by k.

— $(l_{uv})_{a,b} = 1$ if there is an edge in G_n from (uo,a) to (vi,b), and $(l_{uv})_{a,b} = 0$ otherwise;
— $(e_u)_{0,n} = 1$ if there is an edge from (uo,n) to $(E,0)$, and $(e_u)_{a,b} = 0$ otherwise;
— $(s_v)_{0,n} = 1$ if there is an edge from (vo,n) to $(S,0)$, and $(s_v)_{a,b} = 0$ otherwise.

One can simply make sure that no information is lost, the graph G_n – and by transitivity the input n – is totally and faithfully encoded in M_n.

This representation of binary integers is however non-uniform: the size of the matrix depends on the size of the binary representation. This is where the use of von Neumann algebras takes its importance: any matrix algebra can be embedded in the type II_1 hyperfinite factor $\mathfrak{R}$. To get a uniform representation of integers, we therefore only need to embed the matricial representation in $\mathfrak{R}$. Before explaining this step in Section 4, we review in the next section some basics of the theory of von Neumann algebras. The aim of this section is not to introduce the reader to the theory which is much too rich to be condensed here, but to give some ideas and intuitions on it. In the end of the next section, we introduce the crossed product construction, an operation which will be fundamental in the subsequent sections.

3. Von Neumann Algebras and Crossed Products

This section aims at giving a quick overview of the theory of von Neumann algebras. Most of the material it contains is not needed for understanding the results that follow, and the reader can skip this section for a first reading. Section 4 uses the fact that we are working in the type II_1 hyperfinite factor, but the only results it uses is the fact that any matrix algebra can be embedded in a type II_1 factor (Proposition 2), the definition of the crossed product algebra (Definition 5) and some properties of unitary operators acting on a Hilbert space. The remaining sections of the paper do not use results of the theory of operator algebras, except for the last section which contains a technical lemma (Lemma 31) whose proof essentially relies on Theorem 6.

3.1. *Hilbert Spaces and Operators*

We consider the reader familiar with the notions of Hilbert spaces and operators (continuous —or equivalently bounded— linear maps between Hilbert spaces). We refer to the classic textbooks of Conway (1990) for the bases of the theory, and of Murphy (1990) for an excellent introduction to the theory of operator algebras. We will not dwell on the definitions and properties of von Neumann algebras, factors, and hyperfiniteness. We believe all these notions, though used in this paper and in Girard's, are not at the core of the characterization, and will not play an important rôle in the following construction. We therefore refer to the series of Takesaki (2001, 2003*a*, 2003*b*). A quick overview of the needed material can also be found in the appendix of a paper by one the authors (Seiller 2012*b*).

We recall that an operator T is a linear map from $\mathbb{H}$ —a Hilbert space— to $\mathbb{H}$ that is continuous. A standard result tells us that this is equivalent to T being bounded, i.e. that there exists a constant C such that for all $\xi \in \mathbb{H}$, $\|T\xi\| \leqslant C\|\xi\|$. The smallest such constant defines a norm on $\mathscr{L}(\mathbb{H})$ —the set of operators on $\mathbb{H}$—which we will denote by $\|T\|$.

Being given an operator T in $\mathscr{L}(\mathbb{H})$, we can show the existence of its *adjoint* —denoted by T^*—, the operator that satisfies $<T\xi,\eta> = <\xi,T^*\eta>$ for all $\xi,\eta \in \mathbb{H}$. It is easily shown that $T^{**} = T$, i.e. that $(\cdot)^*$ is an involution, and that it satisfies the following conditions:

1 For all $\lambda \in \mathbb{C}$ and $T \in \mathscr{L}(\mathbb{H})$, $(\lambda T)^* = \bar{\lambda} T^*$;
2 For all $S,T \in \mathscr{L}(\mathbb{H})$, $(S+T)^* = S^* + T^*$;
3 For all $S,T \in \mathscr{L}(\mathbb{H})$, $(ST)^* = T^* S^*$.

In a Hilbert space $\mathbb{H}$ there are two natural topologies, the topology induced by the norm on $\mathbb{H}$, and a weaker topology defined by the inner product.

1 The strong topology: we say a sequence $\{\xi_i\}_{i\in\mathbf{N}}$ converges strongly to 0 when $\|\xi_i\| \to 0$.
2 The weak topology: a sequence $\{\xi_i\}_{i\in\mathbf{N}}$ converges weakly to 0 when $<\xi_i,\eta> \to 0$ for all $\eta \in \mathscr{L}(\mathbb{H})$. Weak convergence is thus a point-wise or direction-wise convergence.

On $\mathscr{L}(\mathbb{H})$, numerous topologies can be defined, each of which having its own advantages and drawbacks. The five most important topologies are the norm topology, the strong operator topology, the weak operator topology, the ultra-strong (or σ-strong) topology and the ultra-weak (or σ-weak) topology. We can easily characterize the first three topologies in terms of converging sequences as follows:

1 The norm topology: $\{T_i\}_{i\in\mathbf{N}}$ converges (for the norm) to 0 when $\|T_i\| \to 0$;
2 The strong operator topology, which is induced by the strong topology on $\mathbb{H}$: $\{T_i\}_{i\in\mathbf{N}}$ converges strongly to 0 when, for any $\xi \in \mathbb{H}$, $T_i\xi$ converges strongly to 0 ;
3 The weak operator topology, which is induced by the weak topology on $\mathbb{H}$: $\{T_i\}_{i\in\mathbf{N}}$ converges weakly to 0 when, for any $\xi \in \mathbb{H}$, $T_i\xi$ converges weakly to 0.

We can show that $\mathscr{L}(\mathbb{H})$ is the dual of a space denoted by $\mathscr{L}(\mathbb{H})_*$ containing the *trace-class operators*. For further details, the reader may refer to (Murphy 1990) or (Takesaki 2001). We remind here of this result only to define the σ-weak topology: if A is a topological space and A^* is its dual, the *weak* topology* on A is defined as the point-wise topology.

3.2. *Von Neumann Algebras in a Nutshell*

Let $\mathbb{H}$ be a Hilbert space, and $\mathscr{L}(\mathbb{H})$ be the set of bounded —continuous— linear maps from $\mathbb{H}$ to itself. It is standard knowledge that $\mathscr{L}(\mathbb{H})$ is an associative algebra when endowed with composition and pointwise scalar multiplication and addition. It is moreover a complete normed vector space for the operator norm, defined as $\|u\| = \sup\{x \in \mathbb{H} \mid \|u(x)\|/\|x\|\}$. It is therefore what is called a Banach algebra. On the other hand, it is known that every element of $\mathscr{L}(\mathbb{H})$ has an *adjoint* operator u^*. This operation $(\cdot)^*$ is an involution satisfying $(t+u)^* = t^* + u^*$, $(tu)^* = u^* t^*$, $(\lambda u)^* = \bar{\lambda} u^*$, $\|u^*\| = \|u\|$, and $\|u^* u\| = \|u\|^2$. A Banach algebra endowed with such an involution is called a C*-algebra. As it turns out (this is the famous Gelfand-Naimark-Segal (GNS) construction), any C*-algebra can be represented as a norm-closed $*$-subalgebra of $\mathscr{L}(\mathbb{H})$ for a Hilbert space $\mathbb{H}$.

A von Neumann algebra $\mathfrak{K}$ is a C*-subalgebra of $\mathscr{L}(\mathbb{H})$, where $\mathbb{H}$ is a Hilbert space, which is closed for a weaker topology than the norm topology: the strong-operator topology, which is pointwise convergence on $\mathbb{H}$ considered with its norm topology. The first important result of the theory, obtained by von Neumann, is that this requirement is equivalent to the requirement that $\mathfrak{K}$ is closed for the even weaker *weak operator topology* which is

pointwise convergence on $\mathbb{H}$ considered with its weak topology. It is also equivalent to a completely algebraic condition which is the fact that $\mathfrak{K}$ is equal to its bi-commutant: let us denote $\mathfrak{K}'$ —the commutant of $\mathfrak{K}$— the set of elements of $\mathscr{L}(\mathbb{H})$ which commute with every element of $\mathfrak{K}$, then $\mathfrak{K}''$ denotes the bi-commutant of $\mathfrak{K}$, that is the commutant of the commutant of $\mathfrak{K}$.

The study of von Neumann algebras was quickly reduced to the study of factors, that is von Neumann algebras $\mathfrak{K}$ whose center —the algebra of elements commuting with every element of $\mathfrak{K}$— is trivial: i.e. von Neumann algebras $\mathfrak{K}$ such that $\mathfrak{K} \cap \mathfrak{K}' = \mathbf{C}1_{\mathfrak{K}}$. Indeed, any von Neumann algebra can be decomposed along its center as a direct integral (a continuous direct sum) of factors. Factors $\mathfrak{N}$ can then be easily classified by considering their sets of projections (operators p such that $p = p^* = p^2$). Two projections p, q are *equivalent in* $\mathfrak{N}$ —denoted by $p \sim_{\mathfrak{N}} q$— in the sense of Murray and von Neumann if there exists a partial isometry $u \in \mathfrak{N}$ such that $uu^* = p$ and $u^*u = q$. A projection p is *infinite in* $\mathfrak{N}$ if there exists a proper subprojection $q < p$ (where $q \leqslant p$ is defined as $pq = q$, i.e. as the inclusion of the subspaces corresponding to p and q) such that $p \sim_{\mathfrak{N}} q$. A projection is *finite* when it is not infinite. The classification of factor is as follows:

— **Type I:** $\mathfrak{N}$ contains non-zero finite minimal projections. If the identity of $\mathfrak{N}$ is the sum of a finite number —say n— of minimal projections, $\mathfrak{N}$ is of type I_n, and if it is not the case $\mathfrak{N}$ is of type I_∞.

— **Type II:**$\mathfrak{N}$ contains finite projections but has no minimal projections. Then if the identity of $\mathfrak{N}$ is a finite projection, $\mathfrak{N}$ is of type II_1, and it is of type II_∞ otherwise.

— **Type III:**all the non-zero projections of $\mathfrak{N}$ are infinite.

A typical example of type I_n factor is the algebra of $n \times n$ matrices. Similarly, a typical example of type I_∞ factor is the algebra $\mathscr{L}(\mathbb{H})$ of bounded linear maps from a Hilbert space $\mathbb{H}$ to itself. Examples of type II and type III factors are more difficult to come by, and are generally constructed as von Neumann algebras defined from groups, or as von Neumann algebras induced by the (free and ergodic) action of a topological group acting on a measured space. Both these constructions are particular cases of the crossed product construction which is defined at the end of this section.

Proposition 2. *Any matrix algebra can be embedded in a type* II_1 *factor.*

Proof. Let k be an integer, and $\mathfrak{M}$ denote the algebra of $k \times k$ matrices. Let $\mathfrak{N}$ be a type II_1 factor. One can find in $\mathfrak{N}$ a family $\pi_1, \ldots, \pi_k$ of projections such that $\sum_{i=1}^k \pi_i = 1$ and which are equivalent in the sense of Murray and von Neumann, i.e. there exists partial isometries $(u_{i,j})_{1 \leqslant i < j \leqslant k}$ such that $u_{i,j}u_{i,j}^* = \pi_i$ and $u_{i,j}^* u_{i,j} = \pi_j$. We will denote by $u_{j,i}$ the partial isometry $u_{i,j}^*$ and by $u_{i,i}$ the projection π_i. We can then define an embedding Ψ of $\mathfrak{M}$ into $\mathfrak{N}$ as follows:

$$(a_{i,j})_{1 \leqslant i,j \leqslant k} \mapsto \sum_{i,j} a_{i,j} u_{i,j}$$

One can then easily check that Ψ is a $*$-algebra injective morphism. □

Among von Neumann algebras, the approximately finite dimensional ones are of particular interest, and are usually called *hyperfinite*. These are algebras in which every

operator can be approximated (in the sense of the σ-weak topology[7]) by a sequence of finite-dimensional operators (elements of type I_n factors, for $n \in \mathbf{N}$). In particular, the type II_1 hyperfinite factor is unique up to isomorphism (in fact, hyperfinite factors of almost all types are unique).

The definition we gave of von Neumann algebras is a concrete definition, i.e. as an algebra of operators acting on a Hilbert space. It turns out that von Neumann algebras can be defined abstractly as C*-algebras that are the dual space of a Banach space. In the next subsection, and more generally in this paper, the term 'von Neumann algebra' will have the meaning of 'abstract von Neumann algebra'.

3.3. *von Neumann Algebras and Groups*

Definition 3 (Representations)**.** Let $\mathfrak{K}$ be a von Neumann algebra. A couple $(\mathbb{H}, \rho)$ where $\mathbb{H}$ is a Hilbert space and ρ is a $*$-homomorphism from $\mathfrak{K}$ to $\mathscr{L}(\mathbb{H})$ is a *representation* of $\mathfrak{K}$. If ρ is injective, we say the representation is *faithful*.

Among the numerous representations of a von Neumann algebra, one can prove the existence (Haagerup 1975) of the so-called *standard representation*, a representation satisfying several important properties.

The operation that will be of interest to us will be that of taking the *crossed product* of an algebra and a group. This operation is closely related to that of semi-direct product of groups and is a way of internalizing automorphisms. Given an algebra $\mathfrak{A}$ and a group G of automorphisms of $\mathfrak{A}$, we construct the algebra $\mathfrak{A} \rtimes G$ generated by the elements of $\mathfrak{A}$ and the elements of G.

Definition 4. An action of a topological group G on a von Neumann algebra $\mathfrak{K}$ is a continuous homomorphism of G into $\mathrm{Aut}(\mathfrak{K})$.

Definition 5 (Crossed product (representations))**.** Let $(\mathbb{H}, \rho)$ be a representation of a von Neumann algebra $\mathfrak{K}$, G a locally compact group, and α an action of G on $\mathfrak{K}$. Let $\mathbb{K} = L^2(G, \mathbb{H})$ be the Hilbert space of square-summable $\mathbb{H}$-valued functions on G. We define representations π_α of $\mathfrak{K}$ and λ of G on $\mathbb{K}$ as follows

$$\begin{array}{rcll} (\pi_\alpha(x).\xi)(g) & = & (\rho(\alpha(g)^{-1}(x))\xi(g) & (x \in \mathfrak{K}, \xi \in \mathbb{K}, g \in G) \\ (\lambda(g).\xi)(h) & = & \xi(g^{-1}h) & (g, h \in G, \xi \in \mathbb{K}) \end{array}$$

Then the von Neumann algebra on $\mathbb{K}$ generated by $\pi_\alpha(\mathfrak{K})$ and $\lambda(G)$ is called the crossed product of $(\mathbb{H}, \rho)$ by α.

An important fact is that the result of the crossed product does not depend on the chosen representation of $\mathfrak{K}$. The following theorem, which states this fact, will be of use in a technical lemma at the end of this paper.

Theorem 6 (Unicity of the crossed product (Takesaki 2003*a*, Theorem 1.7, p. 241))**.** *Let $(\mathbb{H}, \rho)$ and $(\mathbb{K}, \rho')$ be two faithful representations of a von Neumann algebra $\mathfrak{K}$, and let G be*

[7] In a nutshell, the algebra $\mathscr{L}(\mathbb{H})$ is the dual of the algebra of *trace-class operators*. As a dual, it thus inherits the traditional weak* topology, which is called in the context of von Neumann algebras the σ-weak topology.

a locally compact group together with an action α on $\mathfrak{K}$. Then there exists an isomorphism between the crossed product of $(\mathbb{H},\rho)$ by α and the crossed product of $(\mathbb{K},\rho')$ by α.

As a consequence, one can define *the* crossed product of a von Neumann algebra and a group acting on it by choosing a particular representation. Of course, the natural choice is to consider the *standard representation*.

Definition 7 (Crossed product)**.** Let $\mathfrak{K}$ be a von Neumann algebra, G a group and α an action of G on $\mathfrak{K}$. The algebra $\mathfrak{K} \rtimes_\alpha G$ is defined as the crossed product of the standard representation of $\mathfrak{K}$ by α.

A particular case of crossed product is the crossed product of $\mathbf{C}$ by a (trivial) action of a group G. The resulting algebra is usually called the group von Neumann algebra $\mathfrak{N}(G)$ of G. As it turns out, the operation of internalizing automorphisms of algebras (the crossed product) and the operation of internalizing automorphisms of groups (the semi-direct product) correspond: the algebra $\mathfrak{N}(G \rtimes_\alpha H)$ is isomorphic to $\mathfrak{N}(G) \rtimes_{\tilde{\alpha}} H$ where $\tilde{\alpha}$ is the action of H on $\mathfrak{N}(G)$ induced by the action of H on G.

4. Integers in the Hyperfinite Factor

4.1. *Binary Representation*

We will embed the $(k+1)\times(k+1)$ matrices of Definition 1 in the hyperfinite factor $\mathfrak{R}$ to have a uniform representation of the integers: an integer will be represented by an operator in $\mathfrak{M}_6(\mathfrak{R})$ fulfilling some properties. To express them we define, given a sequence $\star a_1 \dots a_k$ representing an integer n and for $j,l \in \{0,1\}$, the sets:

$$\begin{aligned} I^n_{jl} &= \{1 \leqslant i \leqslant k \mid a_i = j, a_{i+1} = l\} \\ I^n_{Sj} &= \{i = 1 \mid a_i = j\} \\ I^n_{jE} &= \{i = k \mid a_i = j\} \end{aligned}$$

Roughly speaking, I^n_{Sj} (resp. I^n_{jE}) tells us about the first (resp. last) bit of n and I^n_{jl} is the set of sequences of a j followed by a l.

Definition 8 (Binary representation of integers)**.** An operator $N_n \in \mathfrak{M}_6(\mathfrak{R})$ is a *binary representation* of an integer n if there exists projections $\pi_0, \pi_1, \dots, \pi_k$ in $\mathfrak{R}$ that satisfy $\sum_{i=0}^k \pi_i = 1$ such that:

$$N_n = \begin{pmatrix} 0 & l_{00} & 0 & l_{10} & l_{S0} & 0 \\ l^*_{00} & 0 & l^*_{01} & 0 & 0 & l^*_{0E} \\ 0 & l_{01} & 0 & l_{11} & l_{S1} & 0 \\ l^*_{10} & 0 & l^*_{11} & 0 & 0 & l^*_{1E} \\ l^*_{S0} & 0 & l^*_{S1} & 0 & 0 & 0 \\ 0 & l_{0E} & 0 & l_{1E} & 0 & 0 \end{pmatrix}$$

where the coefficients are partial isometries fulfilling the equations (where $\pi_{k+1} = \pi_0$):

$$\begin{aligned} l_\star &= \sum_{i \in I^n_\star} \pi_{i+1} l_\star \pi_i \quad (\star \in \{00,01,10,11,S0,S1,0E,1E\}) \\ \pi_0 &= (l_{0E} + l_{1E})(l_{00} + l_{01} + l_{10} + l_{11})^{k-1}(l_{S0} + l_{S1}) \end{aligned}$$

Proposition 9 (Binary and matricial representations)**.** *Given $N_n \in \mathfrak{M}_6(\mathfrak{R})$ a binary representation of the integer n, there exists an embedding $\theta : \mathfrak{M}_{k+1}(\mathbf{C}) \to \mathfrak{R}$ such that*[8] *$Id \otimes \theta(M_n) = N_n$, where M_n is the matricial representation of n.*

Proof. Let $N_n \in \mathfrak{R}$ a binary representation of $n \in \mathbf{N}$, and $\pi_0, \ldots, \pi_k$ the associated projections. Notice that the projections π_i are pairwise equivalent.

We now define an embedding $\theta : \mathfrak{M}_{k+1}(\mathbf{C}) \to \mathfrak{R}$:

$$\theta : (a_{i,j})_{0 \leqslant i,j \leqslant k} \mapsto \sum_{i=0}^{k} \sum_{j=0}^{k} a_{i,j} u_{i,j}$$

with:

$$u_{i,j} = \begin{cases} (l_{00} + l_{01} + l_{10} + l_{11})^{j-1}(l_{S0} + l_{S1}) & \text{if } i = 0 \\ (l_{00} + l_{01} + l_{10} + l_{11})^{j-1} & \text{if } i < j \text{ and } i \neq 0 \\ ((l_{00} + l_{01} + l_{10} + l_{11})^{i-1}(l_{S0} + l_{S1}))^* & \text{if } j = 0 \\ ((l_{00} + l_{01} + l_{10} + l_{11})^{i-1})^* & \text{if } i > j \text{ and } j \neq 0 \\ \pi_k & \text{if } i = j = k \end{cases}$$

We can easily check that the image by $\mathrm{Id} \otimes \theta$ of the matrix M_n representing n is equal to N_n. □

This new representation is a gain in terms of uniformity, as all the integers are represented by matrices of the same size. But at the same time, as any embedding $\theta : \mathfrak{M}_{k+1}(\mathbf{C}) \to \mathfrak{R}$ define a representation of the integers, we have to check that they all are equivalent (Proposition 10) and to define (Definition 11) a framework where the representation of the integers and the programs can interact as expected.

Proposition 10 (Equivalence of binary representations)**.** *Given N_n and N'_n two binary representations of $n \in \mathbf{N}$, there exists a unitary $u \in \mathfrak{R}$ such that $(Id \otimes u)N_n(Id \otimes u)^* = N'_n$.*

Proof. Let $\pi_0, \ldots, \pi_n$ (resp. $\nu_0, \ldots, \nu_n$) be the projections and $l_\star$ (resp. $l'_\star$) the partial isometries associated to N_n (resp. N'_n). It is straightforward that π_0 and ν_0 are equivalent according to Murray and von Neumann definition, so there exists a partial isometry v such that $vv^* = \nu_0$ and $v^*v = \pi_0$. For all $0 \leqslant i \leqslant n$ we define the partial isometries:

$$v_i = ((l'_{00} + l'_{01} + l'_{10} + l'_{11})^{i-1}(l'_{S0} + l'_{S1}))v((l_{00} + l_{01} + l_{10} + l_{11})^{i-1}(l_{S0} + l_{S1}))^*$$

We can easily check that:

$$\begin{aligned} v_i v_i^* &= \nu_i \\ v_i^* v_i &= \pi_i \end{aligned}$$

It follows that the sum $u = \sum_{i=0}^{n} v_i$ is a unitary and $(\mathrm{Id} \otimes u)N_n(\mathrm{Id} \otimes u)^* = N'_n$. □

[8] We denote by Id the identity matrix of $\mathfrak{M}_6(\mathbf{C})$. We will allow ourselves the same abuse of notation in the following statements and proofs in order to simplify the formulas.

4.2. *Normative Pairs*

The notion of *normative pair*, a pair of two subalgebras $(\mathfrak{N},\mathfrak{O})$, was defined by Girard (Girard 2012) in order to describe the situations in which an operator in $\mathfrak{O}$ acts uniformly on the set of all representations of a given integer in $\mathfrak{N}$. Indeed, as we just explained, we no longer have uniqueness of the representation of integers. An operator representing a kind of abstract machine should therefore interact in the same way with different representations of the same integer.

The notion of normative pair therefore depends on the notion of interaction one is considering. The interaction used by Girard was based on Fuglede-Kadison determinant[9]. As a matter of fact, Girard defines his interaction with the determinant but actually uses nilpotency in his proofs. In order to give more flexibility to the definitions, we chose to work with an interaction based on nilpotency, which represents the fact the computation ends. This change in definition does not modify the fact that one can characterize **co-NL**, but allows one to consider a broader class of groups[10], and a broader class of languages[11].

Definition 11 (Normative Pairs)**.** Let $\mathfrak{N}$ and $\mathfrak{O}$ be two subalgebras of a von Neumann algebra $\mathfrak{K}$. The pair $(\mathfrak{N},\mathfrak{O})$ is a *normative pair (in $\mathfrak{K}$)* if:

— $\mathfrak{N}$ is isomorphic to $\mathfrak{R}$;
— For all $\Phi \in \mathfrak{M}_6(\mathfrak{O})$ and $N_n, N'_n \in \mathfrak{M}_6(\mathfrak{N})$ two binary representations of n,

$$\Phi N_n \text{ is nilpotent} \Leftrightarrow \Phi N'_n \text{ is nilpotent}$$

Proposition 12. *Let S be a set and for all $s \in S$, $\mathfrak{N}_s = \mathfrak{R}$. For all group G and all action α of G on S, the algebra $\mathfrak{K} = (\bigotimes_{s\in S} \mathfrak{N}_s) \rtimes_{\hat{\alpha}} G$ – where $\hat{\alpha}$ denotes the action induced by α on the tensor product – contains a subalgebra generated by G that we will denote $\mathfrak{G}$. Then for all $s \in S$, the pair $(\mathfrak{N}_s,\mathfrak{G})$ is a normative pair (in $\mathfrak{K}$).*

Proof. From the hypotheses, $\mathfrak{N}_s$ is isomorphic to $\mathfrak{R}$. Regarding the second condition, we will only show one implication, the other being obtained by symmetry. By Lemma 10, there exists a unitary u such that $(\mathrm{Id}\otimes u)N_n(\mathrm{Id}\otimes u)^* = N'_n$. We define $v = \bigotimes_{s\in S} u$ and π_v the unitary in $\mathfrak{K}$ induced by v. Then π_v commutes with the elements of $\mathfrak{G}$, so if there exists $d \in \mathbf{N}$ such that $(\phi N_n)^d = 0$, then $(\phi N'_n)^d = (\phi u N_n u^*)^d = (u\phi N_n u^*)^d = u(\phi N_n)^d u^* = 0$. □

Definition 13 (Observations)**.** Let $(\mathfrak{N},\mathfrak{G})$ be a normative pair. An *observation* is an operator in $\mathfrak{M}_6(\mathfrak{G})\otimes\mathfrak{Q}$, where $\mathfrak{Q}$ is a matrix algebra, i.e. $\mathfrak{Q} = \mathfrak{M}_s(\mathbf{C})$ for an integer s, called the *algebra of states*.

[9] A generalization of the usual determinant of matrices that can be defined in a type II_1 factor.

[10] The use of the determinant forces Girard to consider only amenable groups, so that the result of the crossed product in Proposition 12 yields the type II_1 hyperfinite factor.

[11] In this paper and in Girard's, we consider languages obtained from finite positive linear combinations of unitaries induced by the group elements. The positivity of the coefficients is needed so that the condition involving the determinant implies the nilpotency. However, these conditions are no longer equivalent if one allows negative coeficients. As a consequence, this new definition of normative pair extends the number of languages that can be defined.

Definition 14. Let $(\mathfrak{N}, \mathfrak{G})$ be a normative pair, and ϕ an observation. We define the set of natural numbers:

$$[\phi] = \{n \in \mathbf{N} \mid \phi N_n \text{ is nilpotent, } N_n \text{ a binary representation of } n\}$$

Definition 15. Let$(\mathfrak{N}_0, \mathfrak{G})$ be a normative pair and $X \subset \cup_{i=1}^{\infty} \mathfrak{M}_6(\mathfrak{G}) \otimes \mathfrak{M}_i(\mathbf{C})$ be a set of observations. We define the *language decided by* X as the set:

$$\{X\} = \{[\phi] \mid \phi \in X\}$$

Corollary 16. *Let $\mathfrak{S}$ be the group of finite permutations over* $\mathbf{N}$*, and for all* $n \in \mathbf{N}$*,* $\mathfrak{N}_n = \mathfrak{R}$*. Then* $(\mathfrak{N}_0, \mathfrak{G})$ *is a normative pair in* $(\bigotimes_{n\in\mathbf{N}} \mathfrak{N}_n) \rtimes_{\hat{a}} \mathfrak{S}$*.*

In this particular case, the algebra $(\bigotimes_{n\in\mathbf{N}} \mathfrak{N}_n) \rtimes_{\hat{a}} \mathfrak{S}$ is the type II_1 hyperfinite factor. This is one of the reason why Girard considered it, as it is then possible to use Fuglede-Kadison determinant. From now on, we will consider this normative pair fixed, and we will study two sets of observations.

Definition 17 ($P_{\geqslant 0}$ and P_+)**.** An observation $(\phi_{i,j})_{0\leqslant i,j\leqslant 6s} \in \mathfrak{M}_6(\mathfrak{G}) \otimes \mathfrak{M}_s(\mathbf{C})$ is said to be *positive* (resp. *boolean*) when for all i, j, $\phi_{i,j}$ is a positive finite linear combination (resp. a finite sum) of unitaries induced by elements of $\mathfrak{S}$, i.e. $\phi_{i,j} = \sum_{k\in I_{i,j}} \alpha_{i,j}^k \lambda(\sigma_{i,j}^k)$ with $\alpha_{i,j}^k \geqslant 0$ (resp. $\alpha_{i,j}^k = 1$).

We then define the following sets of observations:

$$\begin{aligned} P_{\geqslant 0} &= \{\phi \mid \phi \text{ is a positive observation}\} \\ P_+ &= \{\phi \mid \phi \text{ is a boolean observation}\} \end{aligned}$$

It is not clear at this point how a program could be expressed as an observation. In the next section, we will introduce a notion of abstract machines which is well suited to be represented as an observation. We will then show how one can define an observation that simulates such a machine.

5. Non-Deterministic Pointer Machines

We define in this section the notion of non-deterministic pointer machines (*NDPM*), an abstract device really close to the multi-head finite automata (Rosenberg 1966), well known to characterize logspace computation. The two have in common the fact that they may only move a fixed number of pointers, read the pointed values and according to their (non-deterministic) transition function change the position of the pointers and their state.

However, we felt it necessary to introduce this model of computation because it has several peculiarities that will help encode them as operators:

— It is 'universally non-deterministic': if one branch of computation rejects, the whole computation rejects. It is convenient because acceptance is represented as the nilpotency of an operator.

— Acceptance and rejection are in the codomain of the transition function, and not states, because we want the computation to stop or to loop immediately, and not to have to define the 'last movement' of the pointers.

— The alphabet is fixed to $\{0,1,\star\}$, because these are the only values encoded in the binary representation of the integers.
— Its input is circular, because in the binary representation we can access both the last and first bits of the integer from the symbol $\star$.
— The 'initial configuration' (in fact, the pseudo-configuration, defined below) is a parameter that will be used to make the operator loop properly.
— The values are read and stored only when the pointer move, because before the computation starts, the operator cannot access the input.
— If the transition relation is not defined for the current situation, the *NDPM* accepts, because that's the way the operator will behave. So acceptation is the 'default' behavior, whereas rejection is meaningfull. We could equivalently have forced the relation transition to be total.

Moreover, we will prove in the following that *NDPMs* can be modified to always halt, and to move at most one pointer at a time.

This device may remind of the programming language PURPLE (Hofmann & Schöpp 2009) as we cannot remember any value nor access the address of the pointers, and it may be interesting to study the relations between the latter and our machines. However, since this paper is focused on the study of a non-deterministic framework[12], we postpone this question to a future work dealing with deterministic complexity classes. Here, we will focus on proving that *NDPMs* can recognize any **co-NL** set.

A pointer machine is given by a set of pointers that can move back and forth on the input tape and read (but not write) the values it contains, together with a set of states. For $1 \leqslant i \leqslant p$, given a pointer p_i, only one of three different *instructions* can be performed at each step:

$$p_i+, \text{ i.e. 'move one step forward'},$$
$$p_i-, \text{ i.e. 'move one step backward'},$$
$$\epsilon_i, \text{ i.e. 'do not move'}$$

We define the set of instructions $I_{\{1,\dots,p\}} = \{p_i+, p_i-, \epsilon_i \mid i \in \{1,\dots,p\}\}$. We will denote by $\sharp p_i$ the value of the pointer (the address it points at), that is the number of cells clockwise between $\star$ and the bit pointed by p_i, i.e. the *distance* between $\star$ and the bit pointed. Note that the alphabet Σ is fixed to $\{0,1,\star\}$.

Definition 18. A non-deterministic pointer machine with $p \in \mathbf{N}^*$ pointers is a pair $M = (Q, \rightarrow)$ where Q is the set of *states* and $\rightarrow \subseteq (\{0,1,\star\}^p \times Q) \times ((I^p_{\{1,\dots,p\}} \times Q) \cup \{\textbf{accept, reject}\})$ is the *transition relation*. We write $NDPM(p)$ the set of *NDPMs* with p pointers.

We define a *pseudo-configuration* c of $M \in NDPM(p)$ as a 'partial snapshot': an element in $\{0,1,\star\}^p \times Q$ that contains the last values read by the p pointers and the current state, *but does not contain the addresses of the p pointers*. The set of pseudo-configurations of a machine M is written C_M and it is the domain of the transition relation.

Let $M \in NDPM(p)$, $c \in C_M$ and $n \in \mathbf{N}$ an input. We define $M_c(n)$ as M with n encoded as a string on its circular input tape (as $\star a_1 \dots a_k$ for $a_1 \dots a_k$ the binary encoding of n and

[12] Since the writing of this paper, an article dealing with a non-deterministic variant of PURPLE has been published (Hofmann, Ramyaa & Schöpp 2013).

$a_{k+1} = a_0 = \star$) starting in the pseudo-configuration c with $\sharp p_i = 0$ for all $1 \leqslant i \leqslant p$ (that is, the pointers are initialized with the address of the symbol $\star$). The pointers may be considered as variables that have been declared but not initialized yet. They are associated with *memory slots* that store the values and are updated only when the pointer moves, so as the pointers did not moved yet, those memory slots haven't been initialized. The *initial pseudo-configuration* c initializes those p registers, not necessarily in a faithful way (it may not reflect the values contained at $\sharp p_i$). The entry n is *accepted* (resp. *rejected*) by M with *initial pseudo-configuration* $c \in C_M$ if after a finite number of transitions every branch of $M_c(n)$ reaches **accept** (resp. at least a branch of M reaches **reject**). We say that $M_c(n)$ halts if it accepts or rejects n and that M decides a set S if there exists an initial pseudo-configuration $c \in C_M$ such that $M_c(n)$ accepts if and only if $n \in S$. We write $\mathscr{L}(M)$ the set decided by M.

Definition 19. Let **{NDPM}** be the class of sets S such that there exists a *NDPM* that decides S.

One movement at a time We can prove that for all $M \in NDPM(p)$ there exists $M \in NDPM(p)$ such that for all $\sigma_1,\dots,\sigma_p,\mathbf{q} \to' p_1,\dots,p_p,\mathbf{q}'$ at most one instruction among $p_1,\dots,p_p$ differs from ϵ_i —stated informally, such that no more than one pointer moves at every transition— and such that $\mathscr{L}(M) = \mathscr{L}(M')$. The number of states of M' and the number of transitions needed by M' to decide the same set increase, but that does not affect our machine in terms of complexity as the number of transitions and the cardinality of Q will not be measures of the complexity of our machines.

Shorthands We use the symbol $*$ for any symbol among $\{0,1,\star\}$, 0/1 for '0 or 1'. For instance $(*,0,\mathbf{q}) \to (\epsilon_1,p_2+,\mathbf{q}')$ will be a shorthand for

$$\begin{aligned}(0,0,\mathbf{q}) &\to (\epsilon_1,p_2+,\mathbf{q}')\\ (1,0,\mathbf{q}) &\to (\epsilon_1,p_2+,\mathbf{q}')\\ (\star,0,\mathbf{q}) &\to (\epsilon_1,p_2+,\mathbf{q}')\end{aligned}$$

Sensing pointers We can easily mimic 'sensing pointers', i.e. answer the question '*Is* $\sharp p_1 = \sharp p_2$*?*', by the help of the following routine, which need a third pointer p_3 with $\sharp p_3 = 0$. At every transition, p_1 and p_2 move one square left and p_3 moves one square right. Two cases arise:

— p_1 and p_2 reach $\star$ after the same transition,

— p_1 (or p_2) reaches $\star$ whereas the other pointer is not reading $\star$.

According to the situation, we encode that they were at the same position or not in the state. Then p_1 and p_2 moves at each transition one square right, p_3 moves at each transition one square left, and when $\sharp p_3 = 0$, we resume the computation. We can easily check that p_1 and p_2 are back to their initial position, and now we can retrieve from the state if they were at the same position or not, i.e. if we had $\sharp p_1 = \sharp p_2$. Notice moreover that p_3 is back to $\star$ and ready to be used for another comparison.

To express any number It is possible to express a distance superior to the size k of the input to a routine: j pointers can represent a distance up to k^j. Every time the i-th pointer made a round-trip (that is, is back on $\star$), the $i+1$-th pointer goes one cell right. By acting like the hands of a clock, the j pointers can encode any integer inferior to k^j.

To decode the distance expressed by j pointers $p_1,\dots,p_j$, it is sufficient to have j pointers $p'_1,\dots,p'_j$ and to move them clockwise until for all $1 \leqslant i \leqslant j$, $\sharp p_i = \sharp p'_i$.

We will for the sake of simplicity consider that any distance $\mathcal{O}(k^j)$ can be expressed by a single pointer, even if it may require several pointers to be properly expressed. We make this idea formal in the proof of Lemma 25, by defining how to implement a clock in any *NDPM*.

Pointer arithmetic It is classical pointer arithmetic to prove that with the help of some additional pointers, *NDPMs* can compute addition, subtractions, multiplication, division, logarithm and modulo, i.e. that given two pointers p_1 and p_2, it is possible to let a third pointer p_3 be at $\sharp p_1+\sharp p_2$, $\sharp p_1-\sharp p_2$, $\sharp p_1\times\sharp p_2$, $\lfloor\sharp p_1/\sharp p_2\rfloor$, $\lceil\log(\sharp p_1)\rceil$ or $\sharp p_1 \bmod \sharp p_2$. Needless to say, those operations permit to establish bit by bit the binary expression of an integer encoded by $\sharp p$.

We will only deal with decision problems and ask ourselves what sets can be recognized in this framework. It turns out that we can recognize any **co-NL**-set, and to prove it we will use the most common method[13]: we will exhibit a *NDPM* that can solve a **co-NL**-complete problem, and define another *NDPM* that reduce any **co-NL** problem to this **co-NL**-complete problem.

Definition 20 (**STConnComp**)**.** We define the following problem: 'given a (directed) graph encoded as a list of adjacences, accept if and only if there is **no** path from the source (numbered 1) to the target (numbered n) in the graph'. This problem, known as **STConnComp** or **REACHABILITYComp**, is **co-NL** complete. We define the set

STConnComp = $\{n \in \mathbf{N} \mid n$ does **not** encode a graph where there is a path from 1 to $n\}$

Proposition 21. ***STConnComp*** $\in$ ***{NDPM}***

Proof. Given a graph of size n, the input will be encoded as

$$\star\underbrace{00\dots00}_{n \text{ bits}}\boxed{1}\overbrace{a_{11}0a_{12}0\dots0a_{1n-1}0a_{1n}}^{\text{edges going from1}}\boxed{1}\dots\boxed{1}\overbrace{a_{n1}0a_{n2}0\dots0a_{nn-1}0a_{nn}}^{\text{edges going from } n}\boxed{1}$$

where (a_{ij}) is the adjacency list, that is to say that $a_{ij} = 1$ if there is an edge from the vertex numbered by i to the vertex numbered by j, 0 elsewhere. The boxed bits in the figure above are 'separating' bits, between the coding of n and the list of adjacences, and between the coding of the edges of source i and the coding of the edges of source $i+1$.

We define a *NDPM* M such that $M_c(n)$ with $c = \{\star,\star,\star,\star,\mathbf{Init}\}$ accepts if and only if $n \in$ **STConnComp**.

[13] That can be reminded to the reader in (Arora & Barak 2009), pp. 88–89.

$$(\star,\star,\star,\star,\textbf{Init}) \to (p_1+,p_2+,p_3+,p_4+,\textbf{Init}) \tag{1}$$
$$(*,0,*,*,\textbf{Init}) \to (\epsilon_1,p_2+,p_3+,\epsilon_4,\textbf{Init}) \tag{2}$$
$$(*,1,*,*,\textbf{Init}) \to (\epsilon_1,p_2+,\epsilon_3,\epsilon_4,\textbf{out.edge?}) \tag{3}$$
$$(*,0,*,*,\textbf{out.edge?}) \to (\epsilon_1,p_2+,\epsilon_3,p_4+,\textbf{no.edge}) \tag{4}$$
$$(*,0,*,*,\textbf{no.edge}) \to (\epsilon_1,\epsilon_2,p_3+,\epsilon_4,\textbf{p3.next.node}) \tag{5}$$
$$(*,1,*,*,\textbf{no.edge}) \to \textbf{accept} \tag{6}$$
$$(*,*,*,*,\textbf{p3.next.node}) \to (\epsilon_1,\epsilon_2,p_3+,\epsilon_4,\textbf{reading.sep.bit}) \tag{7}$$
$$(*,*,0,*,\textbf{reading.sep.bit}) \to (\epsilon_1,\epsilon,p_3+,\epsilon_4,\textbf{p3.next.node}) \tag{8}$$
$$(*,*,1,*,\textbf{reading.sep.bit}) \to (\epsilon_1,p_2+,\epsilon_3,\epsilon_4,\textbf{out.edge?}) \tag{9}$$
$$(*,1,*,*,\textbf{out.edge?}) \to \begin{cases}(\epsilon_1,p_2+,\epsilon_3,p_4+,\textbf{no.edge})\\(p_1+,\epsilon_2,\epsilon_3,p_4+,\textbf{edge.found})\end{cases} \tag{10}$$
$$(*,*,*,1,\textbf{edge.found}) \to \textbf{reject} \tag{11}$$
$$(1,*,*,0,\textbf{edge.found}) \to \textbf{accept} \tag{12}$$
$$(*,*,*,0,\textbf{edge.found}) \to (\epsilon_1,p_2-,\epsilon_3,p_4-,\textbf{rewind.p2.p4}) \tag{13}$$
$$(*,*,*,0/1,\textbf{rewind.p2.p4}) \to (\epsilon_1,p_2-,\epsilon_3,p_4-,\textbf{rewind.p2.p4}) \tag{14}$$
$$(*,*,*,\star,\textbf{rewind.p2.p4}) \to (\epsilon_1,p_2-,\epsilon_3,\epsilon_4,\textbf{rewind.p2}) \tag{15}$$
$$(*,0/1,*,*,\textbf{rewind.p2}) \to (\epsilon_1,p_2-,\epsilon_3,\epsilon_4,\textbf{rewind.p2}) \tag{16}$$
$$(*,\star,*,*,\textbf{rewind.p2}) \to (\epsilon_1,p_2+,p_3-,\epsilon_4,\textbf{exchange.p2.p3.}) \tag{17}$$
$$(*,*,0/1,*,\textbf{exchange.p2.p3}) \to (\epsilon_1,p_2+,p_3-,\epsilon_4,\textbf{exchange.p2.p3.}) \tag{18}$$
$$(*,*,\star,*,\textbf{exchange.p2.p3}) \to (\epsilon_1,\epsilon_2,p_3+,\epsilon_4,\textbf{get.p3.to.start}) \tag{19}$$
$$(*,*,0,*,\textbf{get.p3.to.start}) \to (\epsilon_1,\epsilon_2,p_3+,\epsilon_4,\textbf{get.p3.to.start}) \tag{20}$$
$$(*,*,1,*,\textbf{get.p3.to.start}) \to (\epsilon_1,p_2+,\epsilon_3,\epsilon_4,\textbf{out.edge?}) \tag{21}$$

Figure 1. The transition relation to decide **STConnComp**

The transition relation of M is presented in the figure 1. Informally, our algorithm goes as follow:

The pointer p_1 counts the size of the path followed. Every time we follow an edge, we move p_1 forward on the string made of n bits (second line of 10). The pointer p_2 will scan the encoding of the outgoing edges of a vertex, 'followed' by p_3: when p_2 is reading a_{ij} then p_3 will be at a_{j1}. If $a_{ij} = 1$ (premise of 10), a non-deterministic transition takes place: on one hand we continue to scan the outgoing edges from i, on the other we increment p_1, place p_2 at a_{j1} and p_3 at a_{11}. The pointer p_4 'follows' p_3 on the n first bits, and if p_4 reaches a 1 when p_2 reads that there is an edge, it means that there is an edge whose target is n, and so we reject (11). When p_2 finishes to browse the adjacency list of an edge, we accept (6). If p_1 reaches a 1 and p_4 reads a 0 (premise of 12), it means that we already followed n edges without ever targeting the vertex n, so we end up accepting. As we know

that if there is a path from 1 to n then there exists a path of size at most n, $M_c(n)$ will accept if and only if $n \in \mathbf{STConnComp}$, elsewhere $M_c(n)$ rejects.

□

We now adapt the classical logspace-reduction from any problem in **NL** to **STConn**. Given a **co-NL**-problem **Pb**, there exists a non-deterministic logspace Turing Machine M that decides it. To solve **Pb** is just to establish if there is **no** transition from the initial configuration to a rejecting configuration of M, once the computational graph of M is given. We now make some assumptions on M and prove how a *NDPM* can output any bit of the transition graph of M.

Given any set $\mathbf{Pb} \in \mathbf{co\text{-}NL}$, there exists a non-deterministic logspace Turing Machine M such that M accepts n[14] iff $n \in \mathbf{Pb}$. We can assume w.l.o.g. that M works on the alphabet $\Sigma = \{0, 1\}$, does not cycle, always halts, has one read-only tape and one read-write working tape whose precise bound is $k \times (\log(|n|))$. Those are classical 'hacking' of Turing Machines that should not surprise the reader. We may also assume that the names of the states are written in binary, so that for $|Q| = q$ the number of states of M, any state may be written with $\lceil \log q \rceil$ bits. At last, we may assume that the instructions to move the heads are written with two bits. All those assumptions make clear that M may be entirely described with a binary string.

We know that $M(n)$ has less than

$$2^{(k \times (\log(|n|)))} \times (k \times (\log(|n|))) \times (\log(|n|)) \times \lceil \log(q) \rceil$$

different configurations. It reflects respectively the content of the working tape, the position of the read-write and read-only heads and the state. This is equivalent to $2^{\mathcal{O}(\log(|n|))}$, so we know there exists a d such that $M(n)$ has less than $|n|^d$ different configurations.

Any configuration of $M(n)$ may be described as

$$\underbrace{01000\ldots010\ldots011}_{\text{Position of the read head}}\overbrace{\sigma 0\sigma 0\ldots\sigma 0\sigma 1\sigma 0\ldots\sigma 0\sigma 0\sigma 0\sigma 0\sigma 0\sigma 0\sigma 0}^{\text{Working tape and position of the working head}}\underbrace{01\ldots10}_{\text{state}}$$

where the $\lceil \log(|n|) \rceil$ first bits encode the position of the reading head in binary, σ corresponds to the bits on the working tape and the bit that follows σ equals 1 iff the working head is on that cell. The remaining $\lceil \log(q) \rceil$ bits express the current state.

This binary string is of length $\lceil \log(|n|) \rceil \times (2 \times (\lceil \log |n| \rceil \times k)) \times \lceil \log(q) \rceil$, i.e. there exists a e such that this string is of length inferior to $e \times \log(|n|)^2$.

Among all the binary strings of size $e \times \log(|n|)^2$, some correspond to configurations, and some do not (for instance because the working head is supposed to be in several places at the same time) – we will call them 'phantom configurations'.

The *configuration graph* of M on input n is simply the graph where configurations are vertices, and there is an edge between two vertices iff there is a transition in $M(n)$ between the two corresponding configurations.

Lemma 22 (Pointer-reduction)**.** *For all non-deterministic logspace Turing Machine M,*

[14] Meaning that **all** branches reach **accept** after a finite number of transitions.

there exists a NDPM T *such that for all* n*, given a pointer* p_d *with* $\sharp p_d = j$*,* T *accepts iff the* j*-th bit of the encoding of the computation graph of* M *on input* n *is* 1*, rejects if it is* 0*.*

Proof. Recall we use the encoding of the proof of Proposition 21 to express the encoding of the configuration graph of M. The *NDPM* T will act as a 'transducer' as follows:

— It computes the number of binary strings of size $e \times \log(|n|)^2$. This number is bounded by $2^{e \times \log(|n|)^2}$ and we saw previously that a *NDPM* could express such distances. Then T compares this value to j : if j is inferior, it rejects, if j is equal, it accepts, elsewhere it goes on. This reflects the initial bits set to 0 to express in unary the size of the graph.
— Elsewhere T establishes if j corresponds to a 'separating bit' and accepts or rejects accordingly, that can be simply made with the division and modulo operations.
— Elsewhere, j encodes a query regarding the presence or absence of transition between two configurations a and b. If $a = b$, there is no need to explore this transition[15], and T rejects. Elsewhere T establishes if there is a transition between a and b, and accepts or rejects accordingly.

This last point needs to be made more precise: if $j > 2^{e \times \log(|n|)^2}$ and if j does not correspond to a 'separating bit', it means that the value of j corresponds to the absence or presence of an edge between two vertices. So there exists a and b such that $j = a_{ab}$. A simple arithmetic of pointers allows us to retrieve those two values expressed as integers (i.e. as distances).

Then, they are converted to binary strings: the positions of the read-only heads need a bit of pointer arithmetic to be obtained and compared, but the rest of the integer just needs to be compared bitwise. The rest of the binary expression of the vertex encodes directly the configuration, and as all the transitions make only local changes to them, there is only a constant number of information to remember.

Every time there is a difference between the binary expression of a and the binary expression of b, T checks that the difference between them is legal regarding the transition function of M —that may be encoded in the states of T or may be given as a parameter.

The transducer T also have to check that a and b are not 'phantom configurations' and that j is not 'too big', i.e. does not represent a query on vertices that does not exists. □

Corollary 23. ***co-NL*** $\subseteq$ ***{NDPM}***

Proof. Let **Pb** $\in$ **co-NL**, there exists a non-deterministic logspace Turing machines N that decides **Pb**. Suppose given $n \in \mathbf{N}$, we will compose the *NDPM* M that solves **STConnComp** with the transducer T that computes the graph of $N(n)$.

Every time M has to read a value, it asks T by letting a pointer be on the position j of the value it wants to know. There is some kind of layer of abstraction in this composition, for M goes through the input tape without ever reading the actual values, but asks the values to T, which actually reads n.

We have to make sure that the j of the proof of Proposition 22 can be big enough: what is the size of the encoding of the graph of $N(n)$? We encode it as being of size $2^{e \times \log(|n|)}$,

[15] Because that would imply that there is a transition from a configuration to itself, and so $M(n)$ is stuck in a loop.

i.e. we also take 'phantom configurations' to be vertices. The encoding of this 'completed' graph —for every string of size $e \times \log(|n|)$ is taken to be one of its vertex, even if it is not reachable— is of size $\mathcal{O}(2^{\log(|n|)})^2$, an expression bounded by a power of $|n|$, so we can express it.

We can suppose moreover that there is a transition between the 'phantom configuration' encoded by $000\ldots001$ and the initial configuration, and that there exists a transition between any rejecting configuration and the 'phantom configuration' encoded by$111\ldots111$. This allows us to keep the **STConnComp** algorithm as is, computing only if there is no path from the vertex 1 to the vertex n.

The transducer T can compute the configuration graph of $N(x)$ bit-by-bit and pass it to M which solves **STConnComp**. So $M \circ T(n)$ accepts iff there is no path from 1 to a rejecting configuration in the graph of $N(n)$, i.e. iff $N(n)$ accepts. Hence $\mathbf{Pb} \in \mathbf{NPM}$. □

It turns out that all *NDPMs* cannot be represented as operators. Indeed, Lemma 29 which establishes the equivalence between *NDPMs* and operators needs an additional requirement: acyclicity. However, as we will now show, a language which is decided by a *NDPM* is decided by an acyclic *NDPM*.

Definition 24 (Acyclicity)**.** A *NDPM* M is said to be *acyclic* when for all $c \in C_M$ and all entry $n \in \mathbf{N}$, $M_c(n)$ halts.

Lemma 25. *For all* NDPM M *that decides a set* S *there exists an acyclic* NDPM M' *that decides* S.

Proof. To prove this, we need to prove that for all $n \in \mathbf{N}$ of size $|n|$ and $c \in C_M$ there exists a $c' \in C_{M'}$ such that if $M_c(n)$ does not halt then $M'_{c'}(n)$ rejects, and if $M_c(n)$ accepts (resp. rejects) then $M'_{c'}(n)$ accepts (resp. rejects).

We know that the number of configurations of M – with p pointers – is bounded by $|n|^p \times (3)^p \times |Q|$ that is to say bounded by $\mathcal{O}(|n|^d)$ for d a constant. So we know that if M does more than $\mathcal{O}(|n|^d)$ transitions, it will never halt. To obtain M' we will simply add $d+1$ pointers that will behave like the hands of a clock. The first one moves forward each time we make a transition. Each time the i-th one has travelled through the whole input tape, the $i+1$-th one moves forward. When the last one is back on the beginning of the input tape, M' rejects. It ensures us that M' – which has appart from that the same computational behaviour as M – will halt after at most $\mathcal{O}(|n|^{d+1})$ transitions. We set $p' = p + d + 1$, and for all $\mathbf{q} \in Q$, every time we had in M the transition:

$$(\vec{i}, \mathbf{q}) \to (\vec{m}, \mathbf{q}')$$

we add to $\to' \in M'$ the following set of transitions (for $p+1 \leqslant a < p'$):

$$(\vec{i}, \star, \ldots, \star, \mathbf{q}) \to' (\vec{m}, p_{p+1}+, \ldots, p_{p'}+, \mathbf{q'})$$
$$(\vec{i}, 0/1, \ldots, 0/1, \mathbf{q}) \to' (\vec{m}, p_{p+1}+, \epsilon_{p+2}, \ldots, \epsilon_{p'}, \mathbf{q'})$$
$$(\vec{i}, 0/1, \ldots, 0/1, i_a = \star, 0/1, , \ldots, 0/1, \mathbf{q}) \to' (\vec{m}, \epsilon_{p+1}, \ldots, \epsilon_{a-1}, p_a+, p_{a+1}+, \epsilon_{a+2}, \ldots, \epsilon_{p'}, \mathbf{q'})$$
$$(\vec{i}, 0/1, \ldots, 0/1, \star, \mathbf{q}) \to' \mathbf{reject}$$

Then, for all $c' = (\vec{i}, p_{p+1}, \ldots, p_{p'}) \in C_{M'}$ that does not appear on the left-hand side in the previous set of transitions, we add $c' \to' \mathbf{reject}$.

For all $c = (m_1, \ldots, m_p, \mathbf{q}) \in C_M$ we define $c^t = (m_1, \ldots, m_p, \star, \ldots, \star, \mathbf{q}) \in C_{M'}$.

Now take a pseudo-configuration $c \in C_M$, several cases arise:

— If $M_c(n)$ was halting, it was in less than $\mathcal{O}(|n|^d)$ transitions so $M'_{c^t}(n)$ will have the same behavior.

— If $M_c(n)$ was entering a loop, $M'_{c^t}(n)$ rejects after $\mathcal{O}(|n|^{d+1})$ transitions.

However, since we supposed that M was deciding S, we know there exists a pseudo-configuration $s \in C_M$ such that for all $n \in \mathbf{N}$, $M_s(n)$ halts, hence never enters a loop. As a result, by considering the pseudo-configuration s^t we can see that M' will decide the set S. Moreover it is clear that for all $c' \in C_{M'}$ and all $n \in \mathbf{N}$, $M'_{c'}(n)$ always halt, so M' is acyclic. □

Definition 26. Let **{ANDPM}** be the class of sets S such that there exists an acyclic *NDPM* that decides S.

Proposition 27.

$$\textbf{\textit{co-NL}} \subseteq \textbf{\textit{\{ANDPM\}}}$$

Proof. Corollary 23 shows that **co-NL** ⊆ **{NDPM}**. Moreover, it is clear that **{ANDPM}** ⊆ **{NDPM}** and the preceding lemma shows that **{NDPM}** ⊆ **{ANDPM}**. As a consequence, we have **{NDPM}** = **{ANDPM}** and thus **co-NL** ⊆ **{ANDPM}**. □

6. Encoding Non-Deterministic Pointer Machines

6.1. *Encoding a Machine*

Our aim in this section is to prove (Lemma 29) that for any acyclic *NDPM* M and pseudo-configuration $c \in C_M$, there exists an observation $M_c^\bullet \in \mathfrak{M}_6(\mathfrak{G}) \otimes \mathfrak{Q}_M$ such that for all $N_n \in \mathfrak{M}_6(\mathfrak{N})$ a binary representation of n, $M_c(n)$ accepts if and only if $M_c^\bullet(N_n \otimes 1_{\mathfrak{Q}_M})$ is nilpotent.

We will define $M_c^\bullet$ as an operator of $\mathfrak{M}_6(\mathfrak{G}) \otimes \mathfrak{Q}_M$, where

$$\mathfrak{Q}_M = \underbrace{\mathfrak{M}_6(\mathbf{C}) \otimes \mathfrak{M}_6(\mathbf{C}) \otimes \cdots \otimes \mathfrak{M}_6(\mathbf{C})}_{p \text{ times}} \otimes \mathfrak{M}_s(\mathbf{C})$$

The intuition is that the j-th copy of $\mathfrak{M}_6(\mathbf{C})$ represents a 'memory block' that contains the last value read by the j-th pointer. We will therefore distinguish for each copy of $\mathfrak{M}_6(\mathbf{C})$ a basis $(0o, 0i, 1o, 1i, s, e)$ corresponding to the different values a pointer can read. The last algebra in the tensor product represents a set of states: we will distinguish a basis $Q \cup B$ where Q is the set of states of the machine M and B is an additional set of states needed for the definition of $M_c^\bullet$. To sum up, the distinguished basis of $\mathfrak{Q}_M$ considered will be denoted by tuples $(a_1, \ldots, a_p, \mathbf{q})$. Notice that such a tuple naturally corresponds to a pseudo-configuration when $q \in Q$.

As a consequence of the tensoring of N_n with the unit of the algebra of states, the integer is considered at the same time in every possible pseudo-configuration. As a result, the computation for c a pseudo configuration represented by the sequence $M_c^\bullet(N_n \otimes 1_{\mathfrak{Q}_M})$,

$(M_c^\bullet(N_n \otimes 1_{\mathfrak{Q}_M}))^2, \dots$ somehow simulates all the computations $M_c(n)$ simultaneously. However, the representation of reject cannot be done without considering an initial pseudo-configuration, something that will be explained in the next subsection.

The main difficulty is now to encode the transition relation. In order to do this, we will encode each couple $(c,t) \in \rightarrow$ by an operator $\phi_{c,t}$. The encoding of the transition relation will then correspond to the sum:

$$\rightarrow^\bullet = \sum_{c \in C_M} \sum_{t \text{ s.t. } c \rightarrow t} \phi_{c,t}$$

Before explaining the encoding of basic operations, we first define the projections π_{0o}, π_{0i}, π_{1o}, π_{1i}, π_{start}, π_{end} of $\mathfrak{M}_6(\mathbf{C})$ as the projections onto the subspace generated by the distinguished basis. We moreover define $\pi_{0\cdot} = \pi_{0i} + \pi_{0o}$ and $\pi_{1\cdot} = \pi_{1o} + \pi_{1i}$ to identify the bit currently read without considering if we come from the left (the output of the bit) or the right (the input of the bit).

For the sake of simplicity, we also define the following operators in $\mathfrak{Q}_M$: if c and c' are respectively equal to $(a_1, \dots, a_p, \mathbf{q})$ and $(a'_1, \dots, a'_p, \mathbf{q}')$, we define the partial isometry:

$$(c \rightarrow c') = (a_1 \rightarrow a'_1) \otimes \cdots \otimes (a_p \rightarrow a'_p) \otimes (\mathbf{q} \rightarrow \mathbf{q}')$$

where

$$(p \rightarrow p') = \begin{array}{c} \\ p' \end{array} \begin{array}{c} \begin{array}{ccccc} & & p & & \end{array} \\ \begin{pmatrix} 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ 0 & \dots & 1 & \dots & 0 \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ (p \in \{a_1, \dots, a_p, \mathbf{q}\}) \\ (p' \in \{a'_1, \dots, a'_p, \mathbf{q}'\}) \end{array}$$

For $\mathbf{S}$ a set of states, we will use the notation $(\mathbf{S} \rightarrow a'_i)$ (denoted $(\rightarrow a'_i)$ when $\mathbf{S}$ contains all possible states) for the element that goes from any state in $\mathbf{S}$ to a'_i, which is defined as $\sum_{s \in \mathbf{S}} (s \rightarrow a'_i)$.

A transition that impacts only the values stored in the subset $p_{i_1}, \dots, p_{i_l}$ and the state $\mathbf{q}$ will be denoted by

$$([a_{i_1} \rightarrow a'_{i_1}]_{i_1}; \dots; [a_{i_l} \rightarrow a'_{i_l}]_{i_l}; \mathbf{q} \rightarrow \mathbf{q'}) = u_1 \otimes u_2 \otimes \cdots \otimes u_p \otimes (\mathbf{q} \rightarrow \mathbf{q'})$$

where $u_i = (a_{i_j} \rightarrow a'_{i_j})$ if $\exists j, i = i_j$, $u_i = \mathrm{Id}$ elsewhere, and $\mathbf{q} \rightarrow \mathbf{q}' = \mathrm{Id}$ if $\mathbf{q} = \mathbf{q}'$.

We are now ready to define the operators needed to encode the basic operations of the machine. Considering the von Neumann algebra $\mathfrak{M}_6(\mathfrak{R}) \otimes \mathfrak{Q}_M$ as $\mathfrak{M}_6(\mathbf{C}) \otimes \mathfrak{R} \otimes \mathfrak{Q}_M$, we will define these operators as tensor products $u \otimes v \otimes w$, where $u \in \mathfrak{M}_6(\mathbf{C})$, $v \in \mathfrak{G} \subset \mathfrak{R}$ and $w \in \mathfrak{Q}_M$.

6.2. *Basic Operations*

From now on, we consider given a machine M and a pseudo-configuration $c \in C_M$.

6.2.1. *Move forward (resp. backward) a pointer, read a value and change state.* We want to encode the action 'move forward (resp. backward) the pointer j when we are in the

pseudo-configuration $c = (a_1,\dots,a_p;\mathbf{q})$, read the value a'_j stored at $\sharp p_j$ and change the pseudo-configuration for $c' = (a_1,\dots,a_{j-1},a'_j,a_{j+1},\dots,a_p;\mathbf{q}')$'. Although the operators we are going to define are all parametric in $\mathbf{q}$ and $\mathbf{q}'$, those parameters won't appear in their name for the sake of readability.

We first define two matrices [out] and [in] that will be used to keep only the values that comes next, respectively for the forward and backward move:

$$[\text{out}] = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad [\text{in}] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The reader can refer to Definition 8 to see that the application of one of those two matrix to the input get the desired result.

We then define the operators $\overleftarrow{m_j}$ and $\overrightarrow{m_j}$, that somehow select the j-th pointer thanks to the transposition $\tau_{0,j}$ that exchanges 0 and j and puts it in the right direction:

$$\overleftarrow{m_j} = [\text{out}] \otimes \tau_{0,j} \otimes (\mathbf{q} \to \mathbf{move}_j) \qquad \overrightarrow{m_j} = [\text{in}] \otimes \tau_{0,j} \otimes (\mathbf{q} \to \mathbf{move}_j)$$

Notice that we also changed the state to $\mathbf{move}_j$. Finally, we define the three operators that encode the different actions the machine will make according to which of the three possible values (0, 1, $\star$) the j-th pointer read. Those operators come in two variants, since in case of a backward move we are going to read the output of a bit, whereas a forward move leads to the reading of the input of a bit[16].

$$\overleftarrow{l_{j,0}} = \pi_{0o} \otimes \tau_{0,j} \otimes ([\to \pi_{0o}]_j;\mathbf{move}_j \to \mathbf{q}')$$
$$\overleftarrow{l_{j,1}} = \pi_{1o} \otimes \tau_{0,j} \otimes ([\to \pi_{1o}]_j;\mathbf{move}_j \to \mathbf{q}')$$
$$\overleftarrow{l_{j,\star}} = \pi_{\star o} \otimes \tau_{0,j} \otimes ([\to \pi_{\star o}]_j;\mathbf{move}_j \to \mathbf{q}')$$

We define $\overrightarrow{l_{j,0}}$, $\overrightarrow{l_{j,1}}$ and $\overrightarrow{l_{j,\star}}$ in a similar way by substituting i to o in the previous equations. Those six operators $\{\overleftarrow{l_{j,b}},\overrightarrow{l_{j,b}} \mid b \in \{0,1,\star\}\}$ allow to move forward or backward according to the direction, when the next bit is b and change the state to $\mathbf{q}'$.

To sum up, we encode the backward and forward moves by:

$$\overleftarrow{m_j} + \sum_{b\in\{0,1,\star\}} \overleftarrow{l_{j,b}} \qquad \text{and} \qquad \overrightarrow{m_j} + \sum_{b\in\{0,1,\star\}} \overrightarrow{l_{j,b}}$$

6.2.2. *Accept.* The case of acceptance is especially easy: we want to stop the computation, so every transition $(a_1,\dots,a_n;\mathbf{q}) \to \mathbf{accept}$ will be encoded by 0.

6.2.3. *Reject.* We want the operator to loop to simulate the reject of the machine. Indeed, a rejection must ensure that the resulting operator $M_c^\bullet(N_n \otimes 1_{\Omega_M})$ will not be nilpotent. A

[16] We consider that $\star o$ = start and $\star i$ = end.

first naive attempt:

$$\text{reject}_{\text{naive}} = \text{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \text{Id} \otimes \pi_{reject}$$

shows that it is possible to make the computation loop, as $N_n^d \neq 0$ for all $d \in \mathbf{N}$.

$$((N_n \otimes 1_{\mathfrak{Q}})\text{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \text{Id} \otimes \pi_{reject})^d = (N_n \otimes \pi_{reject})^d = N_n^d \otimes \pi_{reject}$$

However, as $\rightarrow^\bullet$ is built as a sum of the basic operations, $\text{reject}_{\text{naive}}$ appears in it, and so $M^\bullet(N_n \otimes 1_{\mathfrak{Q}_M})$ cannot be nilpotent[17]. This is problematic since we want this operator to be nilpotent in case of acceptance.

So we have to be a little more clever to insure the operator will loop if *and only if* the operator that simulates the reject is reached. To do that, we simply make the operator go back to the chosen pseudo-configuration $c = (a_1, \ldots, a_p; \mathbf{q}_0)$ when it reaches this operator. In this way, if reject was reached after applying the machine with a pseudo-configuration c', we enforce the computation of the machine on c. As a consequence, if the integer was accepted by the machine in state c, the rejection that corresponds to a computation on c' will be temporary: once rejection attained, the computation restarts with pseudo-configuration c and will therefore halt accepting.

To encode this, we add two states to the machine —$\mathbf{back}_j$ and $\mathbf{move\text{-}back}_j$— for each $j = 1, \ldots, p$, and we define:

$$\begin{aligned}
rm_j &= 1 \otimes \tau_{0,j} \otimes (\mathbf{back}_j \rightarrow \mathbf{move\text{-}back}_j) \\
rr_j &= \pi_{0o} + \pi_{1o} \otimes \tau_{0,j} \otimes ([\rightarrow \pi_{0o} + \pi_{1o}]_j; \mathbf{move\text{-}back}_j \rightarrow \mathbf{back}_j) \\
rc_j &= \pi_{start} \otimes \tau_{0,j} \otimes ([\rightarrow a_j]_j; \mathbf{move\text{-}back}_j \rightarrow \mathbf{back}_{j+1}) \quad (1 \leqslant j < p) \\
rc_p &= \pi_{start} \otimes \tau_{0,p} \otimes ([\rightarrow a_p]_p; \mathbf{move\text{-}back}_p \rightarrow \mathbf{q}_0)
\end{aligned}$$

The operator simulating the reject by making the operator loop is then defined as follows:

$$\text{reject}_c = \left(\sum_{j=1}^{p} rm_j + rr_j + rc_j \right) + (\mathbf{reject} \rightarrow \mathbf{back}_0)$$

Definition 28. Let M be a pointer machine, $\rightarrow$ its transition relation and c a configuration. The operator $M_c^\bullet$ is defined as:

$$M_c^\bullet = \rightarrow^\bullet + \text{reject}_c$$

6.3. *First Inclusions*

Lemma 29. *Let M be an acyclic* NDPM*, $c \in C_M$ and $M_c^\bullet$ the encoding we just defined. For all $n \in \mathbf{N}$ and every binary representation $N_n \in \mathfrak{M}_6(\mathfrak{N}_0)$ of n:*

$$M_c(n) \textit{ accepts} \Leftrightarrow M_c^\bullet(N_n \otimes 1) \textit{ is nilpotent}$$

Proof. Let us fix $n \in \mathbf{N}$ and N_n one of its binary representations. Considering the representation of the reject it is clear that if a branch of $M_c(n)$ rejects, the operator $M_c^\bullet(N_n \otimes 1)$ will not be nilpotent, so we just have to prove that if $M_c(n)$ accepts then $M_c^\bullet(N_n \otimes 1)$ is

[17] Remember that $N_n \otimes 1_{\mathfrak{Q}} = N_n \otimes \text{Id}_{\otimes_{n=1}^p \mathfrak{M}_6(\mathbf{C})} \otimes \pi_{reject} + N_n \otimes \text{Id}_{\otimes_{n=1}^p \mathfrak{M}_6(\mathbf{C})} \otimes (1 - \pi_{reject})$.

nilpotent. We prove its reciprocal: let's suppose $M_c^\bullet(N_n \otimes 1)$ is not nilpotent. In this product N_n is given to the operator $M_c^\bullet$ that starts the simulation of the computation of M with input n in every possible pseudo-configuration at the same time. Since the encoding of M takes in argument a pseudo-configuration $c \in C_M$, we know that there exists a j such that $M_c^\bullet(N_n \otimes 1)\pi_j$ is the simulation of $M_c(n)$, but the computation takes place in the other projections too: for $i \neq j$ it is possible that $M_c^\bullet(N_n \otimes 1)\pi_i$ loops where for a d $(M_c^\bullet(N_n \otimes 1))^d \pi_j = 0$. We can correct this behavior thanks to acyclicity: if $M_c^\bullet(N_n \otimes 1)$ is not nilpotent it is because at some point the **reject** state has been reached. After this state of reject is reached (let's say after $r \in \mathbf{N}$ iterations) we know that $M_c^\bullet(N_n \otimes 1)^r \pi_i$ is exactly the simulation of $M_c(n)$. If it loops again, it truly means that $M_c(n)$ rejects. So we just proved that $M_c^\bullet(N_n \otimes 1)$ is not nilpotent if and only if $(M_c^\bullet(N_n \otimes 1))^d \pi_j \neq 0$ for all $d \in \mathbf{N}$. But it is clear that in this case M with pseudo-configuration c rejects the entry n. □

Proposition 30.

$$\textbf{\textit{co-NL}} \subseteq \textbf{\textit{\{ANDPM\}}} \subseteq \{P_+\} \subseteq \{P_{\geqslant 0}\}$$

Proof. The first inclusion is given by Proposition 27. By Lemma 29, we have **{ANDPM}** $\subseteq$ $\{P_+\}$ since the representation $M_c^\bullet$ of a couple (M, c), where M is an acyclic *NDPM* and $c \in C_M$, is obviously in P_+. Moreover, since $P_+ \subset P_{\geqslant 0}$, we have $\{P_+\} \subseteq \{P_{\geqslant 0}\}$. □

7. Positive observations and co-NL

To show that $\{P_{\geqslant 0}\}$ is included in **co-NL**, we will show that the product of a binary representation and an observation in $P_{\geqslant 0}$ is the image of a matrix by an injective morphism. This return from the type II_1 hyperfinite factor to matrix algebras is necessary to prove that we can reduce the nilpotency of an operator to the nilpotency of a matrix, so that a finite machine can decide it. This fact was used by Girard[18], but we felt it needed to be more precisely stated and proved in the following (quite technical) lemma.

Lemma 31. *We consider the normative pair* $(\mathfrak{N}_0, \mathfrak{G})$ *defined in Corollary 16 and denote by* $\mathfrak{K}$ *the algebra* $(\bigotimes_{n\geqslant 0} \mathfrak{R}) \rtimes \mathfrak{S}$. *Let* N_n *be a binary representation of an integer* n *in* $\mathfrak{M}_6(\mathfrak{N}_0)$ *and* $\Phi \in \mathfrak{M}_6(\mathfrak{G}) \otimes \mathfrak{Q}$ *be an observation in* $P_{\geqslant 0}$. *Then there exists an integer* k, *an injective morphism* $\psi : \mathfrak{M}_k(\mathbf{C}) \to \mathfrak{K}$ *and two matrices* $M \in \mathfrak{M}_6(\mathfrak{M}_k(\mathbf{C}))$ *and* $\bar{\Phi} \in \mathfrak{M}_6(\mathfrak{M}_k(\mathbf{C})) \otimes \mathfrak{Q}$ *such that* $Id \otimes \psi(M) = (N_n \otimes 1_{\mathfrak{Q}})$ *and* $Id \otimes \psi \otimes Id_{\mathfrak{Q}}(\bar{\Phi}) = \Phi$.

Proof. We denote by n the integer represented by N_n and $R \in \mathfrak{M}_{6(n+1)}(\mathbf{C})$ its matricial representation. Then there exists a morphism $\theta : \mathfrak{M}_{n+1}(\mathbf{C}) \to \mathfrak{R}$ such that $\mathrm{Id} \otimes \theta(R) = N_n$ by Proposition 9. Composing θ with the inclusion $\mu : \mathfrak{M}_{n+1}(\mathbf{C}) \to \bigotimes_{n=0}^N \mathfrak{M}_{n+1}(\mathbf{C})$, $x \mapsto x \otimes 1 \otimes \cdots \otimes 1$, we get:

$$\mathrm{Id} \otimes (\bigotimes_{n=0}^{N} \theta(\mu(R)) = \bar{N}_n \otimes \underbrace{1 \otimes \cdots \otimes 1}_{N \text{ copies}}$$

[18] Altough this point is not dwelled on, this statement is necessary in (Girard 2012, Proof of Theorem 12.1, p.258).

where $\bar{N}_n$ is the representation of n in $\mathfrak{M}_6(\mathbf{C}) \otimes \mathfrak{R}$ (recall the representation N_n in the statement of the lemma is an element of $\mathfrak{M}_6(\mathbf{C}) \otimes \mathfrak{K}$).

Moreover, since Φ is an observation, it is contained in the subalgebra induced by the subgroup $\mathfrak{S}_N$ where N is a fixed integer, i.e. the subalgebra of $\mathfrak{S}$ generated by $\{\lambda(\sigma) \mid \sigma \in \mathfrak{S}_N\}$. We thus consider the algebra $(\bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})) \rtimes \mathfrak{S}_N$. It is isomorphic to a matrix algebra $\mathfrak{M}_k(\mathbf{C})$: the algebra $\bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})$ can be represented as an algebra of operators acting on the Hilbert space $\mathbf{C}^{N(n+1)}$, and the crossed product $(\bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})) \rtimes \mathfrak{S}_N$ is then defined as a subalgebra $\mathfrak{I}$ of the algebra $\mathscr{L}(L^2(\mathfrak{S}_N, \mathbf{C}^{(n+1)^N})) \cong \mathfrak{M}_{(n+1)^N N!}(\mathbf{C})$. We want to show that $(N_n \otimes 1_{\Omega})$ and Φ are the images of matrices in $\mathfrak{I}$ by an injective morphism ψ which we still need to define.

Let us denote by α the action of $\mathfrak{S}_N$ on $\bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})$. By definition, $\mathfrak{I} = (\bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})) \rtimes \mathfrak{S}_N$ is generated by two families of unitaries:

— $\lambda_\alpha(\sigma)$ where $\sigma \in \mathfrak{S}_N$;
— $\pi_\alpha(x)$ where x is an element of $\bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})$.

We will denote by γ the action of $\mathfrak{S}$ on $\bigotimes_{n=0}^{\infty} \mathfrak{R}$. Then $\mathfrak{K} = (\bigotimes_{n \geqslant 0} \mathfrak{R}) \rtimes \mathfrak{S}$ is generated by the following families of unitaries:

— $\lambda_\gamma(\sigma)$ for $\sigma \in \mathfrak{S}$;
— $\pi_\gamma(x)$ for $x \in \bigotimes_{n \geqslant 0} \mathfrak{R}$.

As we already recalled, Φ is an observation in $P_{\geqslant 0}$ and is thus contained in the subalgebra induced by the subgroup $\mathfrak{S}_N$. Moreover, N_n is the image through θ of an element of $\mathfrak{M}_{n+1}(\mathbf{C})$. Denoting β the action of $\mathfrak{S}_N$ on $\bigotimes_{n=0}^{N} \mathfrak{R}$, the two operators we are interested in are elements of the subalgebra $\mathfrak{J}$ of $\mathfrak{K}$ generated by:

— $\lambda_\beta(\sigma)$ for $\sigma \in \mathfrak{S}_N$;
— $\pi_\beta(\bigotimes_{n=0}^{N} \theta(x))$ for $x \in \bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})$.

We recall that Φ is a matrix whose coefficients are finite positive linear combinations of elements $\lambda_\gamma(\sigma)$ where $\sigma \in \mathfrak{S}_N$, i.e. (denoting by k the dimension of the algebra of states):

$$\Phi = \Big(\sum_{i \in I_{a,b}} \alpha^i_{a,b} \lambda_\gamma(\sigma^i_{a,b}) \Big)_{1 \leqslant a,b \leqslant 6k}$$

We can therefore associate to Φ the matrix $\bar{\Phi}$ defined as $\bar{\Phi} = (\sum_{i \in I_{a,b}} \alpha^i_{a,b} \lambda_\alpha(\sigma^i_{a,b}))_{1 \leqslant a,b \leqslant 6k}$. We will now use the theorem stating the crossed product algebra does not depend on the chosen representation (Theorem 6). The algebra $\bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})$ is represented (faithfully) by the morphism $\pi_\beta \circ \bigotimes_{n=0}^{\infty} \theta$. We deduce from this that there exists an isomorphism from $\mathfrak{I}$ to the algebra generated by the unitaries $\lambda_\beta(\sigma)$ ($\sigma \in \mathfrak{S}_N$) and $\pi_\beta \circ \bigotimes_{n=0}^{\infty} \theta(x)$ ($x \in \bigotimes_{n=0}^{N} \mathfrak{M}_{n+1}(\mathbf{C})$). This isomorphism induces an injective morphism ω from $\mathfrak{I}$ into $\mathfrak{J}$ such that:

$$\begin{aligned} \omega(\pi_\alpha(x)) &= \pi_\beta\Big(\bigotimes_{n=0}^{N} \theta(x)\Big) \\ \omega(\lambda_\alpha(\sigma)) &= \lambda_\beta(\sigma) \end{aligned}$$

We will denote by ι the inclusion map $\bigotimes_{n=0}^{N} \mathfrak{R} \subset \bigotimes_{n=0}^{\infty} \mathfrak{R}$ and υ the inclusion map $\mathfrak{S}_N \subset \mathfrak{S}$. We will once again use the same theorem as before, but its application is not as immediate as it was. Let us denote by $\mathfrak{S}_N \backslash \mathfrak{S}$ the set of the orbits of $\mathfrak{S}$ for the action of $\mathfrak{S}_N$

by multiplication on the left, and let us chose a representant $\bar{\tau}$ in each of these orbits. Recall the set of orbits is a partition of $\mathfrak{S}$ and that $\mathfrak{S}_N \times \mathfrak{S}_N \backslash \mathfrak{S}$ is in bijection with $\mathfrak{S}$. As a consequence, the Hilbert space $L^2(\mathfrak{S}_N, L^2(\mathfrak{S}_N \backslash \mathfrak{S}, \bigotimes_{n=0}^{\infty} \mathbb{H}))$ is unitarily equivalent to $L^2(\mathfrak{S}, \bigotimes_{n=0}^{\infty} \mathbb{H})$. We will therefore represent $\bigotimes_{n=0}^{N} \mathfrak{R}$ on this Hilbert space and show this representation corresponds to π_γ. For each $x \in \bigotimes_{n=0}^{N} \mathfrak{R}$, we define $\rho(x)$ by:

$$\rho(x)\xi(\bar{\tau}) = \gamma(\bar{\tau}^{-1})(\iota(x))\xi(\bar{\tau})$$

This representation is obviously faithful. We can then define the crossed product of this representation with the group $\mathfrak{S}_N$ on $L^2(\mathfrak{S}_N, L^2(\mathfrak{S}_N \backslash \mathfrak{S}, \bigotimes_{n=0}^{\infty} \mathbb{H}))$. The resulting algebra is generated by the operators (in the following, ξ is an element of the Hilbert space $L^2(\mathfrak{S}_N, L^2(\mathfrak{S}_N \backslash \mathfrak{S}, \bigotimes_{n=0}^{\infty} \mathbb{H}))$):

$$\begin{aligned}
\lambda(\nu)\xi(\bar{\tau})(\sigma) &= \xi(\bar{\tau})(\nu^{-1}\sigma) \\
\pi(x)\xi(\bar{\tau})(\sigma) &= \rho(\beta(\sigma^{-1})(x))\xi(\bar{\tau})(\sigma) \\
&= \gamma(\bar{\tau}^{-1})(\gamma(\sigma^{-1})(\iota(x)))\xi(\bar{\tau})(\sigma) \\
&= \gamma(\bar{\tau}^{-1}\sigma^{-1})(\iota(x)))\xi(\bar{\tau})(\sigma) \\
&= \gamma((\sigma\bar{\tau})^{-1})(\iota(x)))\xi(\bar{\tau})(\sigma)
\end{aligned}$$

Through the identification of $L^2(\mathfrak{S}_N, L^2(\mathfrak{S}_N \backslash \mathfrak{S}, \bigotimes_{n=0}^{\infty} \mathbb{H}))$ and $L^2(\mathfrak{S}, \bigotimes_{n=0}^{\infty} \mathbb{H}))$, we therefore get (where $\xi \in L^2(\mathfrak{S}_N, L^2(\mathfrak{S}_N \backslash \mathfrak{S}, \bigotimes_{n=0}^{\infty} \mathbb{H}))$):

$$\begin{aligned}
\lambda(\nu)\xi(\sigma\bar{\tau}) &= \xi(\nu^{-1}\sigma\bar{\tau}) \\
&= \lambda_\gamma(\nu)\xi(\sigma\bar{\tau}) \\
\pi(x)\xi(\sigma\bar{\tau}) &= \gamma((\sigma\bar{\tau})^{-1})(\iota(x)))\xi(\sigma\bar{\tau}) \\
&= \pi_\gamma(\iota(x))\xi(\sigma\bar{\tau})
\end{aligned}$$

Applying theorem 6 we finally get the existence of an injective morphism ζ from $\mathfrak{J}$ into $\mathfrak{K}$ such that:

$$\begin{aligned}
\pi_\beta(x) &\mapsto \pi_\gamma(\iota(x)) \\
\lambda_\beta(\sigma) &\mapsto \lambda_\gamma(\sigma)
\end{aligned}$$

Figure 2 illustrates the situation. We now define $\psi : \mathfrak{I} \to \mathfrak{K}$ by $\psi = \zeta \circ \omega$. Noticing that $N_n = \mathrm{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes (\pi_\gamma(\iota \circ \mu(\bar{N}_n))$, we get:

$$\begin{aligned}
\mathrm{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \psi(M) &= \mathrm{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \psi(\mathrm{Id} \otimes \pi_\alpha(\mathrm{Id} \otimes \mu)(R)) \\
&= \mathrm{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \pi_\gamma(\iota \circ \bigotimes_{n=0}^{N} \theta(\mu(R))) \\
&= \mathrm{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \pi_\gamma(\iota(\bar{N}_n \otimes 1 \otimes \cdots \otimes 1)) \\
&= \mathrm{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \pi_\gamma(\iota \circ \mu(\bar{N}_n)) \\
&= N_n
\end{aligned}$$

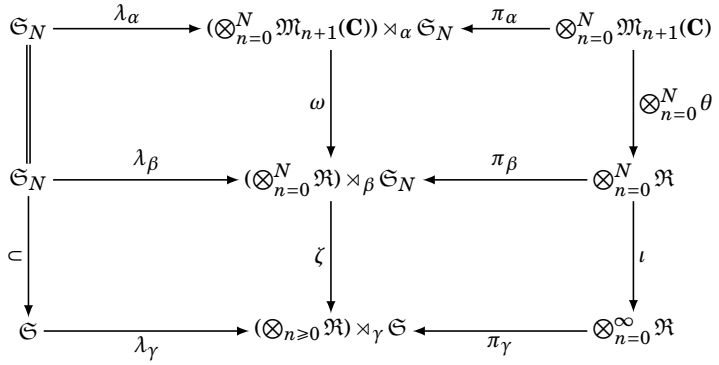


Figure 2. Representation of the main morphisms defined in the proof of Lemma 31

$$\begin{aligned} \mathrm{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \psi \otimes \mathrm{Id}_{\mathfrak{Q}}(\bar{\Phi}) &= (\sum_{i \in I_{a,b}} \alpha_{a,b}^i \psi(\lambda_\alpha(\sigma_{a,b}^i)))_{1 \leqslant a,b \leqslant 6k} \\ &= (\sum_{i \in I_{a,b}} \alpha_{a,b}^i \lambda_\gamma(\sigma_{a,b}^i))_{1 \leqslant a,b \leqslant 6k} \\ &= \Phi \end{aligned}$$

The (injective) morphism ψ thus satisfies all the required properties. □

We are now ready to prove the last inclusion to get the main theorem.

Proposition 32. $\{P_{\geqslant 0}\} \subseteq$ ***co-NL***

Proof. Let $\Phi \in P_{\geqslant 0}$, $\mathfrak{Q}$ its algebra of states and N_n a representation of an integer n. By lemma 31, we know there exists a morphism χ (with ψ as defined in the lemma, $\chi = \mathrm{Id}_{\mathfrak{M}_6(\mathbf{C})} \otimes \psi \otimes \mathrm{Id}_{\mathfrak{Q}}$) and two matrices M and $\bar{\Phi}$ such that $\chi(M \otimes 1_{\mathfrak{Q}}) = N_n \otimes 1_{\mathfrak{Q}}$ and $\chi(\bar{\Phi}) = \Phi$. So we have $\Phi(N_n \otimes 1_{\mathfrak{Q}})$ nilpotent if and only if $\bar{\Phi}(M \otimes 1_{\mathfrak{Q}})$ nilpotent. Our aim is now to prove that checking the nilpotency of this matrix is in **co-NL**.
Our algebra is:

$$\mathfrak{M}_6(\mathbf{C}) \otimes ((\underbrace{\mathfrak{M}_{n+1}(\mathbf{C}) \otimes \cdots \otimes \mathfrak{M}_{n+1}(\mathbf{C})}_{p \text{ copies}}) \rtimes \mathfrak{S}_N) \otimes \mathfrak{Q}$$

and we know an element of its basis will be of the form

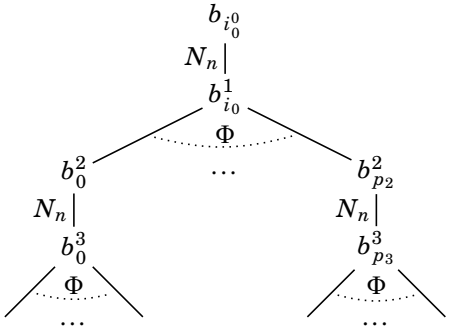
$$(\pi, a_0, a_1, \ldots, a_p; \sigma; e)$$

where π is an element of the basis $(0o, 0i, 1o, 1i, s, e)$ of $\mathfrak{M}_6(\mathbf{C})$, $a_i \in \{1, \ldots, k\}$ (for $i \in \{1, \ldots, p\}$) are the elements of the basis chosen to represent the integer n, $\sigma \in \mathfrak{S}_N$ and e is an element of a basis of $\mathfrak{Q}$. When we apply $M \otimes 1_{\mathfrak{Q}}$ representing the integer to an element of this basis, we obtain one and only one vector of the basis: $(\pi, a_0, a_1, \ldots, a_p; \sigma; e)$. When we apply to this element the observation $\bar{\Phi}$ we obtain a linear positive combination of $L \in \mathbf{N}$ elements of the basis:

$$\bar{\Phi}(\pi, a_0, a_1, \ldots, a_p; \sigma; e) = \sum_{i=0}^{L} \alpha_i (\rho, a_{\tau_i(0)}, \ldots, a_{\tau_i(p)}; \tau_i \sigma; e_i)$$

With a non-deterministic machine, we can follow the computation in parallel on each basis

vector thus obtained. The computation can then be regarded as a tree (denoting by b_i^j the elements of the basis encountered):



We know that L and the nilpotency degree of $\bar{\Phi}(M \otimes 1_{\mathfrak{Q}})$ are both bounded by the dimensions of the underlying space, that is to say $6(k+1)^p p!q$ where q is the dimension of $\mathfrak{Q}$. Since every coefficient α_i is positive, the matrix is thus nilpotent if and only if every branch of this tree is of length at most $6(k+1)^p p!q$.

We only have a logarithmic amount of information to store (the current basis vector), and every time a branch splits a non-deterministic transition takes place to continue the computation on every sub-branch. □

Theorem 33.

$$\textbf{\textit{\{ANDPM\}}} = \{P_+\} = \{P_{\geqslant 0}\} = \textbf{\textit{co-NL}}$$

Proof. By combining Proposition 30 and Proposition 32. □

8. Conclusion and Perspectives

This work explains the motivations and choices made by Girard when he proposed this new approach to study complexity classes. In particular, we explained how the representation of integers by matrices is an abstraction of sequent calculus proofs of the type of binary lists in **ELL**, and how using the hyperfinite factor allows to overcome the lack of uniformity of the matrix representation. We then introduced a notion of normative pair which differs from the one introduced by Girard and showed how the crossed product construction can be used to define such pairs. Going from an interaction based on the determinant to one relying on nilpotency allows to consider a larger class of groups in the construction based on the crossed product. Moreover, even if the two definitions are equivalent in some cases, such as the one considered in this paper, they differ in some others.

We then introduced non-deterministic pointer machines as a technical tool to show that **co-NL** $\subseteq \{P_+\}$. The proof of this inclusion, which was only sketched in Girard's paper, helps to get more insights on how computation is represented by operators. Moreover, it gives a new characterization of **co-NL** in term of machines. We then proved that $\{P_{\geqslant 0}\} \subseteq$ **co-NL** following the proof given by Girard (Girard 2012), providing a proper statement and a proof of the key technical result that was not provided by Girard. Of course, we could have used the famous result which states that **REACHABILITYComp**, is in **NL** (Immerman

1988), to prove that we also characterized **NL**, but we hope to get a different proof of this closure by complementation with our tools.

We believe that this new approach of complexity can be used to characterize other complexity classes. Two different possibilities should be considered: changing the normative pair, and changing the set of observations. As we showed, one could define a normative pair from a group action by using the crossed product construction. However, obtaining new results in this way requires to overcome the difficulty of finding appropriate groups.

The second possibility, which seems at the time less complicated, would be to consider other sets of observations for the same normative pair. For instance, one could define the set of observations whose coefficients are unitaries induced by group elements and whose norm is equal to 1 (so that there are at most one non-zero coefficient in each column). Denoting this set by P_1, we can easily adapt the proof of Proposition 32 to show that $\{P_1\} \subseteq \mathbf{L}$. However, the question of whether the corresponding class $\{P_1\}$ is equal or strictly included in **L**, and its eventual relations to PURPLE (Hofmann & Schöpp 2009), still need to be answered.

References

Arora, S. & Barak, B. (2009), *Computational complexity: a modern approach*, Vol. 1, Cambridge University Press.

Aubert, C. (2011), Sublogarithmic uniform boolean proof nets, *in* J.-Y. Marion, ed., 'DICE', Vol. 75 of *EPTCS*, pp. 15–27.

Baillot, P. & Pedicini, M. (2001), 'Elementary complexity and geometry of interaction', *Fundamenta Informaticae* **45**(1-2), 1–31.

Conway, J. B. (1990), *A course in functional analysis*, Springer.

Dal Lago, U. (2005), The geometry of linear higher-order recursion, *in* 'LICS', IEEE Computer Society, pp. 366–375.

Dal Lago, U. & Hofmann, M. (2010), 'Bounded linear logic, revisited', *Logical Methods in Computer Science* **6**(4), 1–31.

Danos, V. & Joinet, J.-B. (2003), 'Linear logic & elementary time', *Information and Computation* **183**(1), 123–137.

Girard, J.-Y. (1989*a*), 'Geometry of interaction I: Interpretation of system f', *Studies in Logic and the Foundations of Mathematics* **127**, 221–260.

Girard, J.-Y. (1989*b*), Towards a geometry of interaction, *in* 'Proceedings of the AMS Conference on Categories, Logic and Computer Science', pp. 69–108.

Girard, J.-Y. (2011), 'Geometry of Interaction V: Logic in the Hyperfinite Factor', *Theoretical Computer Science* **412**(20), 1860–1883.

Girard, J.-Y. (2012), Normativity in logic, *in* P. Dybjer, S. Lindström, E. Palmgren & G. Sundholm, eds, 'Epistemology versus Ontology', Vol. 27 of *Logic, Epistemology, and the Unity of Science*, Springer, pp. 243–263.

Haagerup, U. (1975), 'The standard form of von neumann algebras', *Mathematica Scandinavia* **37**(271-283).

Hofmann, M., Ramyaa, R. & Schöpp, U. (2013), Pure pointer programs and tree isomorphism, *in* F. Pfenning, ed., 'FoSSaCS', Vol. 7794 of *Lecture Notes in Computer Science*, Springer, pp. 321–336.

Hofmann, M. & Schöpp, U. (2009), Pointer programs and undirected reachability, *in* 'LICS', IEEE Computer Society, pp. 133–142.

Immerman, N. (1988), Nondeterministic space is closed under complementation, *in* 'CoCo', IEEE Computer Society, pp. 112–115.

Lafont, Y. (2004), 'Soft linear logic and polynomial time', *Theoretical Computer Science* **318**(1), 163–180.

Murphy, G. J. (1990), C^**-algebras and operator theory*, Academic Press Inc., Boston, MA.

Rosenberg, A. (1966), 'On multi-head finite automata', *IBM Journal of Research and Development* **10**(5), 388–394.

Schöpp, U. (2007), Stratified bounded affine logic for logarithmic space, *in* 'LICS', IEEE Computer Society, pp. 411–420.

Seiller, T. (2012*a*), 'Interaction graphs: Additives', *Arxiv preprint* **abs/1205.6557**.

Seiller, T. (2012*b*), 'Interaction graphs: Multiplicatives', *Annals of Pure and Applied Logic* **163**, 1808–1837.

Seiller, T. (2012*c*), Logique dans le facteur hyperfini : géometrie de l'interaction et complexité, PhD thesis, Université de la Méditerranée.
URL: *http://tel.archives-ouvertes.fr/tel-00768403/*

Szelepcsényi, R. (1987), 'The method of focing for nondeterministic automata', *Bulletin of the EATCS* **33**, 96–99.

Takesaki, M. (2001), *Theory of Operator Algebras 1*, Vol. 124 of *Encyclopedia of Mathematical Sciences*, Springer.

Takesaki, M. (2003*a*), *Theory of Operator Algebras 2*, Vol. 125 of *Encyclopedia of Mathematical Sciences*, Springer.

Takesaki, M. (2003*b*), *Theory of Operator Algebras 3*, Vol. 127 of *Encyclopedia of Mathematical Sciences*, Springer.

Terui, K. (2004), Proof Nets and Boolean Circuits, *in* 'LICS', IEEE Computer Society, pp. 182–191.