

Interaction Graphs: Nondeterministic Automata*

Thomas Seiller¹

1 Department of Computer Science, University of Copenhagen
Njalsgade 128 bygning 24 (5 sal.), 2300 København S, DENMARK
seiller@di.ku.dk

Abstract

This paper exhibits a series of *semantic* characterisations of sublinear nondeterministic complexity classes. These results fall into the general domain of logic-based approaches to complexity theory and so-called *implicit computational complexity* (ICC), i.e. descriptions of complexity classes without reference to specific machine models. In particular, it relates strongly to ICC results based on linear logic since the semantic framework considered stems from work on the latter. Moreover, the obtained characterisations are of a geometric nature: each class is characterised by a specific action of a group by measure-preserving maps.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes; F.4.1 Mathematical Logic; F.4.3 Formal Languages

Keywords and phrases Implicit Computational Complexity, Semantics, Interaction Graphs, Automata, Linear Logic, Measurable Dynamics

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Complexity theory is concerned with the study of how many resources are needed to perform a specific computation or to solve a given problem. The study of *complexity classes* – sets of problems which need a comparable amount of resources to be solved, lies at the intersection of mathematics and computer science. After the obtention of strong impossibility results [19] preventing the use of known proof methods to settle open separation problems, mathematicians have tried to give characterisations of complexity classes that differ from the original machine-bound definitions, hoping to enable methods from radically different areas of mathematics.

Among them, the field of Implicit Computational Complexity (ICC) aims at studying algorithmic complexity only in terms of restrictions of languages and computational principles. It has been established since Bellantoni and Cook’ landmark paper [6], and following work by Leivant and Marion [16, 17]. Amongst the different approaches to ICC, several results were obtained by considering syntactic restrictions of *linear logic* [11], a refinement of intuitionistic logic which accounts for the notion of resources. Linear logic introduces a modality $!$ marking the “possibility of duplicating” a formula A : the formula A shall be used exactly once, while the formula $!A$ can be used any number of times. Modifying the rules governing this modality then yields variants of linear logic having computational interest: this is how constrained linear logic systems, for instance BLL [14] and ELL [8], are obtained.

Recently, a new line of research emerged, providing semantic characterisations of complexity classes instead of syntactical ones. This approach was initiated by Girard [13] and

* This work was supported by the European Commission’s Marie Skłodowska-Curie Individual Fellowship (H2020-MSCA-IF-2014) 659920 - ReACT.



motivated by his work on Geometry of Interaction (GOI) models, and more precisely the hyperfinit GOI model [12]. Together with C. Aubert, the author showed how Girard's proposal lead to the characterisation of CONLOGSPACE [21, 3] and LOGSPACE [4]. Unfortunately, technical reasons lead the authors to consider modifications of the initial hyperfinite GOI framework, furthering characterisation results from the GOI models construction. In other words, although originating from considerations on semantics, these results were not directly logic-related.

These semantic results were then rephrased in more syntactic terms, providing new characterisations related to logic programming results [1, 2] but taking another step further from the initial framework of the hyperfinite GOI model. After a first step which ended in the loss of an underlying logical framework, this second step ended in the loss of the rich mathematical theories the method was initially founded upon. Although this recent line of work have its own interests, it is the author's belief that one should not forget the mathematical structure from which these characterisations originated. The approach taken in this paper is therefore quite orthogonal to those recent results, as it aims at the obtention of a deeper understanding of how complexity classes can be related to the mathematics behind GOI models in order to provide complexity theorists with new techniques and invariants [26].

Contributions and Outline

The present work achieves three distinct goals related to the logic-based characterisations of complexity classes. Firstly, complexity classes are here characterised as specific types in models of (fragments of) linear logic. It thus fills the gap between the above mentioned series of work GOI-inspired results in computational complexity [3, 4, 1, 2] and the actual semantics provided by GOI models. Secondly, we obtain characterisations of several classes that were not available using previous techniques. This is due to a change of perspective which allows new proof techniques, sensible to more subtle differences. Thirdly, each complexity class considered is here characterised by a specific *group action* on a measure space. This hints at possible uses of mathematical invariants from ergodic theory and measurable group theory in the context of computational complexity.

The paper is constructed as follows. The next section introduces the technical material about *interaction graphs models of linear logic*. This will allow us to define, in section 3, the ambient model which will be used to obtain the characterisations. We also define the representation of binary words and the notion of m_k -machine. Section 4 contains the technical proof of the characterisation: after recalling the definition of multihead automata, we show how the complexity class captured by k -head multihead automata and the one captured by our notion of m_k -machine coincide. Lastly, we discuss this result in the conclusion, providing both a logic and a geometric reading of it.

2 Interaction Graphs Models

2.1 Basic Definitions

Interaction graphs models were introduced by the author in a series of papers [20, 22, 25, 23, 24]. It is a modular framework providing a rich hierarchy of models of (fragments of) linear logic. We describe here the basic operations needed to work out the following section. Proofs are interpreted as a generalisation of graphs, named *graphings*. Graphings can be understood as graphs *realised* on a measured space, i.e. vertices are measurable subsets of the space, and edges represents measurable functions mapping the source subset to the target subset. As

part of the modularity of the framework, we use the notion of *microcosm* to restrict the set of measurable maps the edges of the graphing considered can represent.

► **Definition 1** (Microcosm). Given a measure space $\mathbf{X} = (X, \mathcal{B}, \mu)$, a *microcosm* on $\mathbf{X}$ is a set $\mathfrak{m}$ of non-singular measurable-preserving transformations¹ $\mathbf{X} \rightarrow \mathbf{X}$ which has the structure of a monoid w.r.t. the composition of maps.

In practice, we define microcosms by providing a set of generating maps; this defines a unique microcosm, namely the smallest microcosm containing all given maps.

We must point out that a more general notion of microcosm was introduced in a recent work by the author [24]; the restricted notion defined here is however easier to grasp and sufficient for our purposes. We now define the notion of graphing. Notice that while previous work defined graphing with weights in an arbitrary monoid Ω , we here fix Ω as $[0, 1] \times \{0, 1\}$ with usual multiplication on the unit interval and the product on $\{0, 1\}$. To simplify notations, we write elements of the form $(a, 0)$ as a and elements of the form $(a, 1)$ as $a \cdot 1$.

► **Definition 2** (Graphing representative). We fix a measure space $\mathbf{X}$, a microcosm $\mathfrak{m}$, and a finite set D^G . A *m-graphing representative* G of support V^G and *dialect* D^G is a countable set $\{(S_e^G, i_e^G, o_e^G, \phi_e^G, \omega_e^G) \mid e \in E^G\}$ of 5-tuples where S_e^G is a measurable subset of $V^G \times D^G$, ϕ_e^G is an element of $\mathfrak{m}$, i_e^G, o_e^G are elements of D^G , and $\omega_e^G \in [0, 1] \times \{0, 1\}$ is a *weight*. We will refer to the indexing set E^G as the set of *edges*. For each edge $e \in E^G$ the set $S_e^G \times \{i_e^G\}$ is called the source of e , and we define the *target* of e as the set $T_e^G \times \{o_e^G\}$ where $T_e^G = \phi_e^G(S_e^G)$.

► **Remark.** Most graphings considered in this paper do not use weights different from 1, except for the *tests*, defined in ???. We will therefore allow ourselves to define graphings without mentioning the weights, implying that those are all equal to 1.

Now, graphing representatives are identified w.r.t. almost-everywhere equality and a notion of *refinement*. Formally, a graphing representative F is a refinement of a graphing representative G if there exists a partition² $(E_e^F)_{e \in E^G}$ of E^F such that:

$$\begin{aligned} \forall e \in E^G, \cup_{f \in E_e^F} S_f^F &= S_e^G; & \forall e \in E^G, \forall f \neq f' \in E_e^F, \mu(S_f^F \cap S_{f'}^F) &= 0; \\ \forall e \in E^G, \forall f \in E_e^F, \omega_f^F &= \omega_e^G & \forall e \in E^G, \forall f \in E_e^F, \phi_f^F &= \phi_e^G \end{aligned}$$

Then two graphing representatives are *equivalent* if and only if they possess a common refinement. The actual notion of *graphing* is then an equivalence class of the objects just defined w.r.t. this equivalence. Since all operations considered on graphings were shown to be compatible with this quotienting [25], i.e. well defined on the equivalence classes, we will in general make no distinction between a graphing – as an equivalence class – and a graphing representative belonging to this equivalence class.

Moreover, graphings are shown equivalent w.r.t. dialect-renaming, i.e. if G is obtained from F by renaming the dialect then F and G are *universally equivalent*, i.e. indistinguishable in the model. Consequently, we will always consider in the following that dialects are chosen as initial segments of the natural numbers, i.e. sets $[n] = \{0, 1, \dots, n\}$.

¹ We recall that a *non-singular transformation* is a measurable map f such that $\mu(f(A)) = 0$ if and only if $\mu(A) = 0$; it is *measurable-preserving* when $f(A) \in \mathcal{B}$ whenever $A \in \mathcal{B}$.

² We allow the sets E_e^F to be empty.

2.2 Paths and Execution

We can now define a notion of *alternating* path between two graphings F, G as a sequence of edges $\pi = e_1, e_2, \dots, e_k$ verifying the following two conditions.

- e_i in E^G if and only if $e_{i+1} \in E^F$, and $\circ_{e_i} = \mathbf{i}_{e_{i+2}}$;
- every measurable set $(\phi_{e_i} \circ \phi_{e_{i-1}} \circ \dots \circ \phi_{e_1})(S_{e_1})$ is of strictly positive measure³.

We denote $\text{AltPath}(F, G)$ the set of such paths. A given path naturally represents the composition $\phi_\pi = \phi_{e_k} \circ \dots \circ \phi_{e_1}$ which belongs to $\mathfrak{m}$ since the latter is a monoid. We define the source π as $S_\pi \times \{\mathbf{i}_{e_1}, \mathbf{i}_{e_2}\}$, where S_π is defined as the set of all x such that for all i , $\phi_{e_i} \circ \dots \circ \phi_{e_1}(x) \in S_{e_{i+1}}$. The weight ω_π of the path is defined as $\prod_{i=1}^k \omega_{e_i}$.

Given a path π and a measurable subset C , we define $[\pi]_o^C(C)$ as the path representing the same map as π , and whose source has been restricted to $S^{\vee C} \times \{\mathbf{i}_{e_1}, \mathbf{i}_{e_2}\}$, with $S_\pi^{\vee C} = (S_\pi \cap \bar{C} \cap (\phi_\pi)^{-1}(\bar{C}))$ where $\bar{C}$ is the complement set of C . Intuitively, we restrict π to the subset of its domain that lies outside of C and is mapped outside of C by the map ϕ_π .

► **Definition 3** (Execution). Let F and G be graphings with $V^F = V \uplus C$ and $V^G = C \uplus W$. Their *execution* $F :: G$ is the graphing of support $V \uplus W$ and dialect $D^F \times D^G$ defined as the set of all $[\pi]_o^C(C)$ where π is an alternating path between F and G .

$$F :: G = \{ (S_\pi^{\vee C}, (\mathbf{i}_{e_1}, \mathbf{i}_{e_2}), (\circ_{e_{n-1}}, \circ_{e_n}), \phi_\pi, \omega_\pi) \mid \pi = e_1, e_2, \dots, e_n \in \text{AltPath}(F, G) \}$$

Execution represents the cut-elimination procedure or, through the proofs-as-programs correspondence, the execution of programs. Together, graphings and execution provide dynamic semantics for proofs / programs.

2.3 Orthogonality and Models

The second step in defining interaction graphs models consists in building an interpretation of types as (particular) sets of proof interpretations. This construction builds on a particular case of the (tight, orthogonality) double-glueing construction defined by Hyland and Schalk [15].

We first define the *measurement* between two graphings, and then use it to define a binary relation between graphings – the *orthogonality*. Although the definition of measurement is quite involved in the general case [25], it will be enough for our purpose to consider:

- its restriction to measure-preserving maps;
- a fixed parameter map $m : \Omega \rightarrow \mathbf{R}_{\geq 0} \cup \{\infty\}$ equal to $m(x, y) = xy$;

The measurement is defined by a sum over all *circuits* between two graphs F and G . A *circuit* is an equivalence class of cycles w.r.t. cyclic permutations. The actual sum is computed by a choice of representatives of circuits, i.e. a set $\text{Rep}(F, G)$ of cycles $\pi = (e_i)_{i=0}^{n-1}$ such that (1) $\pi^k \in \text{Rep}(F, G)$ (k a non-zero integer) implies $\pi \in \text{Rep}(F, G)$; (2) $\pi \in \text{Rep}(F, G)$ implies $(e_{i+k})_{i=0}^{n-1} \notin \text{Rep}(F, G)$ (where $i+k$ is computed in $\mathbf{Z}/n\mathbf{Z}$). More details about the definition, and a proof that the considered measurement is independent from this choice of representative is found in previous work by the author [25].

► **Definition 4.** The measurement between two graphings (realised by measure-preserving maps) is defined as

$$[F, G] = \sum_{\pi \in \text{Rep}(F, G)} \int_{\text{supp}(\pi)} \frac{\omega(\pi)^{\rho_{\phi_\pi}(x)}}{\rho_{\phi_\pi}(x)} d\lambda(x)$$

where $\rho_{\phi_\pi}(x) = \inf\{n \in \mathbf{N} \mid \phi_\pi^n(x) = x\}$ (by convention, $\inf \emptyset = \infty$).

³ In particular, $S_{e_{i+1}} \cap T_{e_i}$ is non-negligible.

We now describe the model. For technical reasons explained in previous papers [20, 22], a proof is interpreted as a pair of a real number and a formal weighted sum of graphings of a fixed support – a *sliced graphing*. The measurement and the execution are extended to these objects as follows:

$$\left\| (a, \sum_{i \in I} \alpha_i A_i), (b, \sum_{j \in J} \beta_j B_j) \right\| = a \left(\sum_{j \in J} \beta_j \right) + b \left(\sum_{i \in I} \alpha_i \right) + \sum_{(i,j) \in I \times J} \alpha_i \beta_j \llbracket A_i, B_j \rrbracket$$

$$(a, \sum_{i \in I} \alpha_i A_i) :: (b, \sum_{j \in J} \beta_j B_j) = \left(\left\| (a, \sum_{i \in I} \alpha_i A_i), (b, \sum_{j \in J} \beta_j B_j) \right\|, \sum_{(i,j) \in I \times J} \alpha_i \beta_j A_i :: B_j \right)$$

► **Definition 5.** A project of support V is a pair (a, A) of a real number a and a finite formal sum $A = \sum_{i \in I} \alpha_i A_i$ where for all $i \in I$, $\alpha_i \in \mathbf{R}$ and A_i is a graphing of support V .

► **Definition 6.** Two projects (a, A) and (b, B) are orthogonal – written $(a, A) \perp (b, B)$ – when they have equal support and $\llbracket (a, A), (b, B) \rrbracket \neq 0, \infty$. We also define the orthogonal of a set A as $A^\perp = \{(b, B) : \forall (a, A) \in A, (a, A) \perp (b, B)\}$ and write $A^{\perp\perp}$ the double-orthogonal $(A^\perp)^\perp$.

Based on this orthogonality relation, we can define the notion of *conducts* and *behaviours* which are the interpretations of types in the models.

► **Definition 7.** A *conduct* is a set of projects $\mathbf{A}$ such that $\mathbf{A} = \mathbf{A}^{\perp\perp}$. A *behaviour* is a conduct such that whenever (a, A) belongs to $\mathbf{A}$ (resp. $\mathbf{A}^\perp$) and for all $\lambda \in \mathbf{R}$, then $(a, A + \lambda \emptyset)$ belongs to $\mathbf{A}$ (resp. $\mathbf{A}^\perp$) as well. If both $\mathbf{A}$ and $\mathbf{A}^\perp$ are non-empty, we say $\mathbf{A}$ is *proper*.

In this paper, we will deal almost exclusively with proper behaviours. Based on the following proposition, we will therefore consider mostly projects of the form $(0, L)$ which we abusively identify with the underlying sliced graphing L . Moreover, we will use the term “behaviour” in place of “proper behaviour”.

► **Proposition 8** ([22, Proposition 60]). *If $\mathbf{A}$ is a proper behaviour, $(a, A) \in \mathbf{A}$ implies $a = 0$.*

Finally, let us mention the fundamental theorem for the interaction graphs construction in the restricted case we just exposed⁴.

► **Theorem 9** ([25, Theorem 1]). *For any microcosm $\mathfrak{m}$, the set of behaviours provides a model of Multiplicative-Additive Linear Logic (MALL) without multiplicative units.*

This theorem can be refined, as the set of conducts provides a model of Multiplicative Linear Logic (MLL), although multiplicative units are *not* behaviours. Moreover MALL is only the minimal fragment one can expect to model, and one can define models which interpret second-order quantification [25] as well as exponential connectives [23, 24].

3 Integers, Machines, Tests

3.1 General Situation

We consider a sufficiently large model in order to work with representations of binary words – or integers. Since we will use the Elementary Linear Logic encoding, explained in the next section, we will start by defining a model of ELL. Complexity classes will then be characterised

⁴ The general construction allows for other sets of weights as well as whole families of measurements [25].

as subsets of the type of predicates over binary lists described by a restriction on their *realisers*. In other words, we will consider sets of graphings of a given type that use a fixed submonoid of the microcosm. Alternatively, this submonoid can be understood as defining a specific submodel of linear logic with very weak exponential connectives.

► **Definition 10** (The Space). We will work on the measure space $\mathbf{X} = \mathbf{Z} \times [0, 1]^{\mathbf{N}}$ considered with its usual Borel σ -algebra and Lebesgue measure.

Borrowing the notation introduced in earlier work [24], we denote by $(x, \mathbf{s})$ the points in $\mathbf{X}$, where $\mathbf{s}$ is a sequence for which we allow a concatenation-based notation, i.e. we write $(a, b) \cdot \mathbf{s}$ the sequences whose first two elements are a, b . Given a permutation σ over the natural numbers, we write $\sigma(\mathbf{s})$ the result of its natural action on the $\mathbf{N}$ -indexed list $\mathbf{s}$.

► **Definition 11** (Microcosms). For all integer $i \geq 1$, we consider the microcosm m_i generated by the translations $t_z : (x, \mathbf{s}) \mapsto (x + z, \mathbf{s})$ for all integer z , and the permutations $p_\sigma : (x, \mathbf{s}) \mapsto (x, \sigma(\mathbf{s}))$ for all permutation σ such that $\sigma(k) = k$ for all $k > i$. We write m_∞ the union $\cup_{i \geq 1} m_i$.

We also define the microcosms $\tilde{m}_i$ as the smallest microcosm containing m_i and all translations⁵ $t_\lambda : (x, a \cdot \mathbf{s}) \mapsto (x, (a + \lambda) \cdot \mathbf{s})$ for λ in $[0, 1]$.

We now define a bijective measure-preserving pairing function: $[\cdot, \cdot] : [0, 1]^2 \rightarrow [0, 1]$. Although it will not be used in the next sections, this will help us draw the connection between the present results and models of Elementary Linear Logic.

Given a subset A of $\mathbf{X}$, integers $d < n$, we define the set:

$$\text{push}_d^n(A) = \{(a, [x, y] \cdot \mathbf{s}) : (a, \mathbf{s}) \in A, d \leq nx \leq d + 1, y \in [0, 1]\}.$$

Given a measurable map $f : A \rightarrow B$ and integers $d, d' < n$, we define the measurable map:

$$\text{push}_{d, d'}^n(f) : \begin{cases} \text{push}_d^n(A) & \rightarrow & \text{push}_{d'}^n(B) \\ (a, x \cdot \mathbf{s}) & \mapsto & (a', y \cdot \mathbf{s}') \end{cases} \quad \text{where } (a', \mathbf{s}') = f(a, \mathbf{s}) \text{ and } y = x + (d' - d)/n$$

► **Definition 12.** Given a graphing $G = \{(S_e^G, i_e^G, o_e^G, \phi_e^G)\}$ of dialect $D = [n]$, we define the promotion $!G$ of G as the following graphing of dialect $[0]$:

$$\{(\text{push}_{i_e^G}^n(S_e^G), 0, 0, \text{push}_{i_e^G, o_e^G}^n(\phi_e^G)) \mid e \in E^G\}$$

This previous definition is a *perennisation*, as defined in earlier papers [21, 23], i.e. it maps arbitrary graphings to graphings with trivial dialect $[0]$. Since one can find a graphings C such that $C :: A \simeq A \otimes A$ for all A with a trivial dialect [23, Proposition 36], this implies that all graphings of the form $!A$ are *duplicable*.

► **Definition 13.** Given a behaviour $\mathbf{A}$, we define the conduct $!\mathbf{A}$ as the set $\{(0, !G) \mid G \in \mathbf{A}\}^{\perp\perp}$.

► Remark. It is important to note that the conduct $!\mathbf{A}$ *never* is a behaviour. However, if $\mathbf{B}$ is an arbitrary behaviour, $!\mathbf{A} \multimap \mathbf{B}$ is a behaviour [23, Corollary 57].

► **Theorem 14.** Consider the microcosm $\mathfrak{p}$ generated by $\tilde{m}_\infty$ together with the additional maps pair and pair^{-1} , where $\text{pair} : (a, (x, y) \cdot \mathbf{s}) \mapsto (a, [x, y] \cdot \mathbf{s})$. For any microcosm containing $\mathfrak{p}$, the set of conducts and behaviours is a model of Elementary Linear Logic.

⁵ We denote here by $a + \lambda$ the fractional part of the sum $a + \lambda$.

Proof. We only need to check that functorial promotion can be implemented, as contraction is automatically satisfied [23] and the fact that it is a model of MALL follows from Theorem 9. The technique is similar as what is used in previous papers [23, 24]. First, we notice the maps $\text{inl} = (a, [x, [y, z]] \cdot \mathbf{s}) \mapsto (a, [[x, y], z] \cdot \mathbf{s})$ and $\text{inr} = (a, [x, [y, z]] \cdot \mathbf{s}) \mapsto (a, [[y, x], z] \cdot \mathbf{s})$ belong to the microcosm $\mathbf{p}$. Then, given $F \in \mathbf{A} \multimap \mathbf{B}$ and $A \in \mathbf{A}$, we can check that $\text{inl}(!F) :: \text{inr}(!A)$ is equivalent to $!(F :: A)$, which is an element of $!\mathbf{B}$. $\blacktriangleleft$

3.2 Representation of Integers

We use here the ELL encoding of binary words, i.e. as elements of the type $\text{BList} = \forall X, !(X \multimap X) \multimap !(X \multimap X) \multimap !(X \multimap X)$. We write $\Sigma = \{0, 1\}$, and denote by $\Sigma_\star$ the extended alphabet $\Sigma \cup \{\star\}$: a binary word $\mathfrak{w}$ will be represented with a starting symbol $\star$, i.e. $\mathfrak{w} = \star a_1 a_2 \dots a_n$ where $a_i \in \Sigma$.

► Notations 15. We write Σ^{ll} the set $\Sigma_\star \times \{\text{in}, \text{out}\}$. We also denote by $\Sigma_{a,r}^{\text{ll}}$ the set $\Sigma^{\text{ll}} \cup \{a, r\}$, where a (resp. r) stand for accept (resp. reject).

► Notations 16. We fix once and for all an injection Ψ from the set $\Sigma_{a,r}^{\text{ll}}$ to intervals in $\mathbf{R}$ of the form $[k, k+1]$ with k an integer. For all $f \in \Sigma_{a,r}^{\text{ll}}$ and Y a measurable subset of $[0, 1]^{\mathbf{N}}$, we denote by $\langle f \rangle_Y$ the measurable subset $\Psi(f) \times Y$ of $\mathbf{X}$. If $Y = [0, 1]^{\mathbf{N}}$, we omit the subscript and write $\langle f \rangle$. The notation extends to any subset S of $\Sigma_{a,r}^{\text{ll}}$, i.e. $\langle S \rangle$ is the (disjoint) union $\bigcup_{f \in S} \langle f \rangle$.

Given a word $\mathfrak{w} = \star a_1 a_2 \dots a_k$, we denote $\bar{W}_{\mathfrak{w}}$ the graph with set of vertices $V^{\bar{W}_{\mathfrak{w}}} \times D^{\bar{W}_{\mathfrak{w}}}$, set of edges $E^{\bar{W}_{\mathfrak{w}}}$, source map $s^{\bar{W}_{\mathfrak{w}}}$ and target map $t^{\bar{W}_{\mathfrak{w}}}$ respectively defined as follows:

$$\begin{aligned} V^{\bar{W}_{\mathfrak{w}}} &= \Sigma^{\text{ll}} & s^{\bar{W}_{\mathfrak{w}}} &= (r, i) \mapsto (a_i, \text{out}, i) \\ D^{\bar{W}_{\mathfrak{w}}} &= [k] & & (l, i) \mapsto (a_i, \text{in}, i) \\ E^{\bar{W}_{\mathfrak{w}}} &= \{r, l\} \times [k] & t^{\bar{W}_{\mathfrak{w}}} &= (r, i) \mapsto (a_{i+1}, \text{in}, i+1 \bmod k+1) \\ & & & (l, i) \mapsto (a_{i-1}, \text{out}, i-1 \bmod k+1) \end{aligned}$$

This graph is the discrete representation of $\mathfrak{w}$. Detailed explanations on how these graphs relate to the proofs of the formula BList can be found in earlier work [21, 3].

► **Definition 17.** Let $\mathfrak{w}$ be a word $\mathfrak{w} = \star a_1 a_2 \dots a_k$ over the alphabet Σ . We define the word graphing $W_{\mathfrak{w}}$ of support $\langle \Sigma^{\text{ll}} \rangle$ and dialect $D^{\bar{W}_{\mathfrak{w}}}$ by the set of edges $E^{\bar{W}_{\mathfrak{w}}}$ and for all edge e :

$$\{(\langle f \rangle, i, j, \phi_{f,i}^{g,j}, 1) : e \in E^{\bar{W}_{\mathfrak{w}}}, s^{\bar{W}_{\mathfrak{w}}}(e) = (f, i), t^{\bar{W}_{\mathfrak{w}}}(e) = (g, j), \phi_{f,i}^{g,j} : (\langle f \rangle, x, i) \mapsto (\langle g \rangle, x, j)\}$$

► Notations 18. We write $\text{Gp}(\mathfrak{w})$ the set of word graphings for $\mathfrak{w}$. It is defined as the set of graphings obtained by renaming the dialect $D^{\bar{W}_{\mathfrak{w}}}$ w.r.t. an injection $[k] \rightarrow [n]$.

► **Definition 19.** Given a word $\mathfrak{w}$, a *representation* of $\mathfrak{w}$ is a graphing $!L$ where L belongs to $\text{Gp}(\mathfrak{w})$. The set of representations of words in Σ is denoted $\sharp \mathbf{W}_2$.

We then define the conduct $!\mathbf{Nat}_2 = (\sharp \mathbf{W}_2)^{\perp \perp}$.

► **Definition 20.** We define the (unproper) behaviour $\mathbf{NBool}$ as $\mathbf{T}_{\langle a, r \rangle}$, where for all measurable set V the behaviour $\mathbf{T}_V$ is defined as the set of all projects of support V . For all microcosm $\mathfrak{m}$, we define $\mathbf{Pred}(\mathfrak{m})$ as the set of $\mathfrak{m}$ -graphings in $!\mathbf{Nat}_2 \multimap \mathbf{NBool}$.

3.3 Predicate Machines and Tests

We now turn to the notion of machine. We focus in this paper on machines computing predicates, i.e. elements of the type $!\mathbf{Nat}_2 \multimap \mathbf{NBool}$. Computing devices are traditionally discrete and finite objects, and it is therefore quite natural to envision them as graphs. However, the notion

we consider – called m-machines – will be *realisations* of graphs as m-graphings, i.e. infinite objects in some ways. Intuitively, the underlying graph corresponds to the simple notion of automaton (with the dialect playing the role of control states), while the realisations of edges correspond to particular *instructions*. This intuitive understanding of m-machines can be followed through the rest of this paper.

► **Definition 21.** A graphing G is *finite* when there exists a graphing H such that⁶ $G \leq H$ and the set of edges E^H is finite.

► **Definition 22.** A *nondeterministic predicate m-machine* over the alphabet Σ is a finite m-graphing belonging to **Pred(m)** with all weights equal to 1.

The computation of a given machine given an argument is represented by the *execution*, i.e. the computation of paths defined in subsection 2.2. The result of the execution is an element of **NBool**, i.e. in some ways a generalised boolean value⁷.

► **Definition 23** (Computation). Let M be a m-machine, $w \in \Sigma^*$ and $!L \in !\mathbf{Nat}_2$. The *computation* of M over $!L$ is defined as the graphing $M :: !L$, an element of **NBool**.

We now introduce the notion of *test*. This notion is essential as it allows for considering several notions of acceptance. Even though acceptance may be defined “by hand” by describing directly the expected result, the definition through tests allows for a more interesting definition. Indeed, the acceptance is described inside the model, using already existing notions, i.e. we do not modify the models to define testing. In other words, acceptance and rejection are given a logical meaning, as testing is tied with the process of constructing types.

► **Definition 24** (Tests). A *test* is a family $\mathcal{T} = \{(t_i, T_i) \mid i \in I\}$ of projects of support $\langle a, r \rangle$.

We now want to define the language characterised by a machine. For this, one could consider *existential* $\mathcal{L}_{\exists}^{\mathcal{T}}(M)$ and *universal* $\mathcal{L}_{\forall}^{\mathcal{T}}(M)$ languages for a machine M w.r.t. a test $\mathcal{T}$:

$$\begin{aligned} \mathcal{L}_{\exists}^{\mathcal{T}}(M) &= \{w \in \Sigma^* \mid \forall T_i \in \mathcal{T}, \exists W_w \in \mathbf{Rep}^{(\cdot)}(w), M :: W_w \perp T_i\} \\ \mathcal{L}_{\forall}^{\mathcal{T}}(M) &= \{w \in \Sigma^* \mid \forall T_i \in \mathcal{T}, \forall W_w \in \mathbf{Rep}^{(\cdot)}(w), M :: W_w \perp T_i\} \end{aligned}$$

We now introduce the notion of *uniformity*, which describes a situation where both definitions above coincide. This collapse of definitions is of particular interest because it ensures that both of the following problems are easy to solve:

- whether a word belongs to the language: from the existential definition one only needs to consider one representation of the word;
- whether a word does not belong to the language: from the universal definition, one needs to consider only one representation of the word.

► **Definition 25** (Uniformity). Let m be a microcosm. The test $\mathcal{T}$ is said *uniform* w.r.t. m-machines if for all such machine M , and any two elements $!W, !W'$ for $W, W' \in \mathbf{Gp}(w)$:

$$M :: !W \in \mathcal{T}^{\perp} \text{ if and only if } M :: !W' \in \mathcal{T}^{\perp}$$

Given a m-machine M , we write in this case $\mathcal{L}^{\mathcal{T}}(M) = \mathcal{L}_{\exists}^{\mathcal{T}}(M) = \mathcal{L}_{\forall}^{\mathcal{T}}(M)$.

⁶ We use the notation $G \leq H$ for “ G is a refinement of H ” for the notion of refinement explained in subsection 2.1.

⁷ If one were working with “deterministic machines” [26], it would belong to the subtype **Bool** of booleans.

4 Characterising a nondeterministic Hierarchy

4.1 Multihead Automata

We consider a variant of the classical notion of two-way multihead finite automata obtained by:

- fixing the right and left end-markers as both being equal to the fixed symbol $\star$;
- fixing once and for all unique initial, accept and reject states;
- choosing that each transition step moves exactly one of the multiple heads of the automaton;
- imposing that all heads are repositioned on the left end-marker before accepting/rejecting.

It should be clear that these choices in design have no effect on the sets of languages recognised.

► **Definition 26.** A *two-way multihead automaton* M with k heads is defined as a tuple $(\Sigma, Q, \rightarrow)$, where $\rightarrow \subseteq (\Sigma_{\star}^k \times Q) \times (\{1, \dots, k\} \times \{\text{in}, \text{out}\}) \times Q$ is the *transition relation* of M . The automaton M is *deterministic* when the relation $\rightarrow$ is functional.

The set of two-way multihead automata with k heads is written $\mathbf{2nfa}(k)$, and the set of all two-way multihead automata $\cup_{k \geq 1} \mathbf{2nfa}(k)$ is denoted by $\mathbf{2nfa}$.

► **Definition 27.** We denote $\text{CO2NFA}(k)$ the set of languages accepted by automata in $\mathbf{2nfa}(k)$, where an automaton M accepts a word w if and only there are no computation trace of M given w as input leading to a rejecting state.

The set of languages $\text{REGULAR} = \text{CO2NFA}(1)$ is usually called the set of *regular languages*. We now state two of the main results in the theory of two-way multihead automata.

► **Theorem 28** (Monien [18]). *For all k , the set $\text{CO2NFA}(k)$ is a strict subset of $\text{CO2NFA}(k+1)$.*

► **Theorem 29.** $\cup_{k \geq 1} \text{CO2NFA}(k) = \text{CONLOGSPACE}$.

We will now show how k -head multihead automata corresponds to m_k -machines. Space constraints prevent us from providing examples, but the reader will find some of those in an overview and perspective paper by the author [26].

4.2 Automata as Machines

There are two main differences between the model of multihead automata with k heads and the notion of m_k -machines.

- The first difference is that when one “moves the i -th head” of a m_k -machine, it induces a reindexing of the sets of heads. I.e. a m_k -machine should be understood as a multihead automata that can only move its principal head, but has the possibility of reindexing its heads following any permutation over k elements. To deal with this, we will extend the set of states Q of the automaton we wish to represent and consider $\bar{Q} = Q \times \mathfrak{S}_k$; the set of permutations $\mathfrak{S}_k$ being used to keep track of the heads’ reindexings.
- The second difference comes from the fact that the computation of m_k -machines is “dynamic”, i.e. corresponds to a dialogue between the machine and the representation of the word it is given as input. As a consequence, one has the knowledge of what symbol a given pointer is reading at a given location *only at the exact moment the pointer moves onto this location*. I.e. the pointer receives information about the input from the integer, and one has to store it if it is to be reused later on. This is different from the way multihead automata compute since the latter can, at any given time, access the value located where any head is pointing at. To take care of this difference, we extend once again the set of states. As a consequence, the automaton M with a set of states Q will be realised as a m_k -machine with an extended set of states (encoded as the *dialect*) $\bar{Q} \times \{\star, 0, 1\}^k$.

► **Definition 30.** Let M be an automaton with k heads. We here write $M = (\Sigma, Q, \rightarrow)$. We define $\{M\}$ a graphing in m_k with dialect – set of states – $Q \times \mathfrak{G}_k \times \{\star, 0, 1\}^k$ as follows. The set of edges of $\{M\}$ is the set:

$$E^{\{M\}} = \{(\{t\}, a, d, \sigma) \mid t \in \rightarrow, a \in \{\star, 0, 1\}, d \in \{\text{in}, \text{out}\}, \sigma \in \mathfrak{G}_k\}$$

The source of the edge $(\{t\}, s, d, \sigma)$ for $t = ((\vec{s}, q), (i, d', q'))$ is defined as:

$$S_{(\{t\}, a, d, \sigma)} = \begin{cases} \langle\langle s, d \rangle\rangle \times \langle\langle q, \sigma, \vec{s} \rangle\rangle & \text{if } q \neq \text{init} \\ \langle a \rangle \times \langle\langle \text{init}, \text{Id}, \vec{\star} \rangle\rangle & \text{if } q = \text{init} \text{ and } d = \text{in} \\ \langle r \rangle \times \langle\langle \text{init}, \text{Id}, \vec{\star} \rangle\rangle & \text{if } q = \text{init} \text{ and } d = \text{out} \end{cases}$$

The target of the edge $(\{t\}, s, d, \sigma)$ for $t = ((\vec{s}, q), (i, d', q'))$ is defined as:

$$T_{(\{t\}, a, d, \sigma)} = \begin{cases} \langle\langle s_i, d' \rangle\rangle \times \langle\langle q', \tau_{1, \sigma(i)} \circ \sigma, \vec{s}[s_{\sigma^{-1}(1)} := s] \rangle\rangle & \text{if } q' \notin \{\text{accept}, \text{reject}\} \\ \langle q' \rangle \times \langle\langle \text{init}, \text{Id}, \vec{\star} \rangle\rangle & \text{if } q' \in \{\text{accept}, \text{reject}\} \end{cases}$$

The realiser of the edge $(\{t\}, a, d, \sigma)$ for $t = ((\vec{s}, q), (i, d', q'))$ is the map $p_{(1, \sigma(i))}$ composed with the adequate translation on $\mathbf{Z}$. E.g. when $q \neq \text{init}$ and $q' \notin \{\text{accept}, \text{reject}\}$ it is the map $p_{(1, \sigma(i))}$ composed with the bijection exchanging $\langle\langle s, d \rangle\rangle$ and $\langle\langle s_i, d' \rangle\rangle$.

Let us explain how this encoding simulates the automaton. We fix a word $w = \star a_1 a_2 \dots a_n$ and a configuration C of a k -head automaton, i.e. a sequence of heads positions $(p_i)_{i=1}^k$ – where for all i , $p_i \in \{0, \dots, n\}$ –, and a state q . Depending on the value $\vec{s} = a_{p_1}, \dots, a_{p_k}$, the automaton will fire different transitions. Let us pick one, namely $t = (\vec{s}, q) \rightarrow (i, d', q')$. There is a family of corresponding edges in the automaton, denoted by $(\{t\}, a, d, \sigma)$. Here, σ is a permutation that remembers how heads have been reindexed since the initial transition; as explained above, this is because moving a head requires a reindexing. The pair (s, d) records a symbol and a direction, namely the symbol and direction of the previous transition made by the automaton: it is therefore uniquely fixed when considering a given computation trace. Then a given edge $(\{t\}, a, d, \sigma)$ maps the set $\langle\langle s, d \rangle\rangle \times \langle\langle q, \sigma, \vec{s} \rangle\rangle$ to $\langle\langle s_i, d' \rangle\rangle \times \langle\langle q', \tau_{1, \sigma(i)} \circ \sigma, \vec{s}[s_{\sigma^{-1}(1)} := a] \rangle\rangle$ (supposing $q \neq \text{init}$ and $q' \neq \text{accept}, \text{reject}$). In doing so, it is updating the value of the sequence $\vec{s}$ according the value read by the pointer moved *during the previous transition which lead to (s, d)* . It is also positioning its i th head adequately by reindexing it using the map $p_{(1, \sigma(i))}$ and waiting for the integer to provide its next value in direction d' by fixing the target subset $\langle\langle s_i, d' \rangle\rangle$ (s_i being the last value read by the i -th head).

The following proposition is then proved by induction.

► **Proposition 31.** *Let M be a k -heads automaton. Alternating paths of odd length between $\{M\}$ and $!W_w$ of source $\langle a \rangle_Y$ (resp. of source $\langle r \rangle_Y$) with⁸ $Y = [0, \frac{1}{|\mathfrak{g}(w)|}]^k \times [0, 1]^N$ are in bijective correspondence with the non-empty computation traces of M given w as input.*

► **Corollary 32.** *The automaton M accept the word w if and only if there exists no alternating path between $\{M\}$ and $!W_w$ from $\langle r \rangle$ to itself.*

► **Definition 33.** We define the test $\mathcal{T}_\zeta$ as the set consisting of the projects $t_\zeta^- = (\zeta, \text{Id}_{\langle r \rangle})$, where $\zeta \neq 0$ and $\text{Id}_{\langle r \rangle}$ is the graphing with a single edge and trivial dialect $[0]: \{\langle r \rangle, 0, 0, x \mapsto x, 1 \cdot \mathbf{1}\}$.

The fact that this test is uniform comes from the invariance of the underlying graphing $\text{Id}_{\langle r \rangle}$ w.r.t. any bijective transformation. In more details, two representations of the same integer

⁸ To understand where the subset Y comes from, we refer the reader to the proof of Lemma 39.

$!W$ and $!W'$ can be shown to relate through a measurable (though not measure-preserving) bijection θ by conjugation, i.e. $\phi \mapsto \theta^{-1}\phi\theta$ maps edges in $!W$ to edges in $!W'$. Then, one just has to remark that the realiser of an alternating path between $!W'$ and $\text{Id}_{\langle r \rangle}$ contains subsequences of the form $\theta \circ \theta^{-1}$ which shows, by simplification, that there exists a corresponding path alternating between $!W$ and $\text{Id}_{\langle r \rangle}$.

► **Proposition 34.** *The test $\mathcal{T}_-$ is uniform w.r.t. $\mathfrak{m}_\infty$ -machines.*

► **Theorem 35.** *Let M be a $2\text{nfa}(k)$, w a word. Then $w \in \mathcal{L}^{\mathcal{T}_-}(\{M\})$ if and only if M accepts w .*

Proof. From Proposition 31 and the constraint on automata that they should reinitialise their pointer to the left end-marker before accepting or rejecting, we know that $R = \{M\} :: !W_w$ contains exactly as many edges from $\langle r \rangle_Y$ to $\langle r \rangle_Y$ – here Y is defined as in the statement of Proposition 31 – as there are rejecting computation traces of M given w as input.

Moreover, $\llbracket \{M\}, !W_w \rrbracket_m$ is equal to 0 as all weights of these graphings are equal to 1. Then the result of the computation $(0, R)$ is orthogonal to $\mathcal{T}_-$ if and only if $\xi + \llbracket R, \{\text{Id}_{\langle r \rangle}\} \rrbracket_m \neq 0, \infty$ for all $\xi \neq 0$. Now, this is true if and only if that $\llbracket R, \text{Id}_{\langle r \rangle} \rrbracket_m = 0$, i.e. if and only if there are no edges from $\langle r \rangle_Y$ to $\langle r \rangle_Y$ in R since any such edge creates a cycle with $\text{Id}_{\langle r \rangle}$ of weight $1 \cdot 1$. ◀

► **Theorem 36.** *Any language computed by an acyclic k -head automaton is computed by a $\mathfrak{m}_k$ -machine w.r.t. $\mathcal{T}_-$.*

4.3 Machines as Automata

We will now describe how one can define a i -head automaton computing the same language as any $\mathfrak{m}_i$ -machine. For this purpose, we will first restrict our attention to *essential graphings*; i.e. graphings whose edges are realised by specific maps that correspond to a single instruction. Although the translation could be defined on general $\mathfrak{m}_i$ -machines, this restriction will help ease the formalisation.

► **Definition 37.** A $\mathfrak{m}$ -machine M is Γ -essential w.r.t. a generating set Γ of the microcosm $\mathfrak{m}$ if every edge $e \in E^M$ is realised by a restriction of a map in Γ .

► **Theorem 38.** *Let Γ be a set of measurable maps, $\mathfrak{m}$ the microcosm generated by Γ , and M a $\mathfrak{m}$ -machine. There exists a Γ -essential $\mathfrak{m}$ -machine $\tilde{M}$ such that, for all test $\mathcal{T}$, $\mathcal{L}^{\mathcal{T}}(M) = \mathcal{L}^{\mathcal{T}}(\tilde{M})$.*

Proof. The proof is technical but not difficult. The principle is the following: one considers an extended dialect and then decomposes each edge that is not realised by an element of Γ by a series of edges using specific new states (i.e. newly added elements of the dialect) and going back and forth on the input with the currently active head to stall the computation. ◀

The following is a technical lemma that uses some particular properties of the microcosm $\mathfrak{m}_\infty$. This lemma is the equivalent, on our framework, to the so-called technical lemma which was essential in previous work involving operator algebras [3, 4].

► **Lemma 39** (Technical Lemma). *Let M be a $\mathfrak{m}_\infty$ -machine. The computation of M with a representation $!W$ of a word w is the realisation by translations of a Ω -weighted finite graph.*

Proof. The proof of this lemma is based on the finiteness of $\mathfrak{m}_\infty$ -machines. Since M is a finite graphing, there exists an integer N such that M is a $\mathfrak{m}_N$ -machine. We are thus left to prove the result for M a $\mathfrak{m}_N$ -machine. We now pick a word $w \in \Sigma^*$, write k the length of w and $(0, W_w)$ the project $(0, !W_w)$. Let us remark that all maps realising edges in M or in $!W_w$ are of the form $\phi \times \text{Id}_{\times_{i=N+1}^\infty [0,1]}$. We can therefore consider that the underlying space is $\mathbb{Z} \times [0,1]^N$

instead of $\mathbf{X}$ by just replacing realisers $\phi \times \text{Id}_{\times_{i=N+1}^\infty [0,1]}$ by ϕ . Moreover, the maps ϕ here act either as permutations over copies of $[0, 1]$ (realisers of edges of M) or as permutations over a decomposition of $[0, 1]$ into k intervals (realisers of $!W_{\text{iv}}$). Consequently, all realisers act as permutations over the set of N -cubes $\{\times_{i=1}^N [k_i/k, (k_i+1)/k] \mid 0 \leq k_i \leq k-1\}$, i.e. their restrictions to N -cubes are translations.

Consequently, one can build two (thick⁹) graphs $\bar{M}$ and $\bar{W}_{\text{iv}}$ over the set of vertices $\Sigma^{\text{ll}} \times \{\times_{i=1}^N [k_i/k, (k_i+1)/k] \mid 0 \leq k_i \leq k-1\}$ as follows. There is an edge in $\bar{M}$ of source $(s, (k_i)_{i=1}^N, d)$ to $(s', (k'_i)_{i=1}^N, d')$ if and only if there is an edge in M of source $\langle s \rangle \times \{d\}$ and target $\langle s' \rangle \times \{d'\}$ whose realisation send the N -cube $\times_{i=1}^N [k_i/k, (k_i+1)/k]$ onto the N -cube $\times_{i=1}^N [k'_i/k, (k'_i+1)/k]$. There is an edge in $\bar{W}_{\text{iv}}$ of source $(s, (k_i)_{i=1}^N, d)$ to $(s', (k'_i)_{i=1}^N, d')$ if and only if $d = d'$, $k_i = k'_i$ for $i \geq 2$ and there is an edge in W_{iv} of source $\langle s \rangle \times [k_1/k, (k_1+1)/k] \times [0, 1]^N$ and target $\langle s' \rangle \times [k'_1/k, (k'_1+1)/k] \times [0, 1]^N$.

Then, checking the existence of an alternating path between M and $!W_{\text{iv}}$ turns out to be equivalent to the existence of an alternating path between $\bar{M}$ and $\bar{W}_{\text{iv}}$. ◀

This lemma will be useful because of the following proposition.

► **Proposition 40.** *For any m-machine G and word representation $!W$, $G :: !W$ is orthogonal to $\mathcal{T}_-$ if and only if there are no cycles between G and $!W \otimes \text{Id}_{\langle r \rangle}$ going through $\langle r \rangle$.*

Proof. We use here the *trefoil property* for graphings [25], which in this case translates as $\llbracket (0, G) :: (0, !W), (t, T) \rrbracket_m = \llbracket (0, G), (0, !W) :: (t, T) \rrbracket_m$. Since the support of $!W$ and the test are disjoint, we have the equality $(0, !W) :: (t, T) = (0, !W) \otimes (t, T)$. Hence $G :: !W$ is orthogonal to $\mathcal{T}_-$ if and only if G is orthogonal to $(0, !W) \otimes (\zeta, \text{Id}_{\langle r \rangle})$ for all $\zeta \neq 0$. But $(0, !W) \otimes (\zeta, \text{Id}_{\langle r \rangle}) = (\zeta, !W \otimes \text{Id}_{\langle r \rangle})$. Thus $G :: !W$ is orthogonal to $\mathcal{T}_-$ if and only if $\zeta + \llbracket G, !W \otimes \text{Id}_{\langle r \rangle} \rrbracket_m \neq 0, \infty$, i.e. if and only if there are no alternating cycles between G and $!W \otimes \text{Id}_{\langle r \rangle}$ of weight of the form $\alpha \cdot \mathbf{1}$. Finally, since all weights in G and $!W$ are equal to 1, such cycles need to go through $\langle r \rangle$. ◀

Using these results, we can show the wanted inclusion (i.e. completeness of the model). For this we consider a Γ_N -essential $\mathfrak{m}_N$ -machine G where Γ_N is the subset of $\mathfrak{m}_N$ in which all permutation-induced transformations are of the form $p_{\tau_{i,j}}$ where $\tau_{i,j}$ denotes the transposition exchanging 1 and j . We then construct an automaton $[G]$ that computes the same language as G . We will build the automaton so that it follows the alternating paths between M and $!W \otimes \text{Id}_{\langle r \rangle}$ starting in $\langle r \rangle$, using the fact that this can be done by following the paths between finite graphs $\bar{M}$ and $\bar{W}_{\text{iv}} \otimes \text{Id}$ using Lemma 39.

We construct the automaton $[G]$ as follows. Let Q denote the dialect of the thick graphing M . We denote by $\mathcal{J}$ the set of vertices $(\langle r \rangle, q)$, with $q \in Q$, which are both a source and a target of edges in M . Any cycle going through $\langle r \rangle$ will go through at least one element of $\mathcal{J}$. Notice however, that such a cycle may go through several elements of $\mathcal{J}$, i.e. the cycle may go through the test several times before reaching its initial vertex.

If $\mathcal{J}$ is empty, then $\mathcal{L}^{\mathcal{T}}(M) = \emptyset$ which is clearly computed by an automaton with at most N heads. We now suppose that $\mathcal{J} \neq \emptyset$. We will build an automaton $[G]$ whose set of states is equal to $Q \times \mathfrak{G}_N \times \mathcal{J} \times \{\star, 0, 1\}^N$. The permutations in $\mathfrak{G}_N$ will be used to keep track of the exchanges of heads during the computation. The sequences in $\{\star, 0, 1\}^N$ will be used to remember the starting positions of the heads: indeed a cycle has to go back not only to its initial state but to its initial heads' positions as well.

Notice that the choice of an element of q of the dialect together with a sequence in $\{\star, 0, 1\}^N$ corresponds to the choice of a vertex in the graph $\bar{G}$. Notice also that all edges are realised by

⁹ Thick graphs are graphs with dialects, where dialects act as they do in graphings, i.e. as control states.

a transposition $p_{\tau_{1,j}}$ composed with a bijection on $\mathbf{N}$; we abusively say that the edge is realised by the transposition to lighten the definition of the automaton.

We now define the transition relation of the automaton.

- Each edge e in M , of source $(\langle x \rangle, q)$ with $q \in \mathcal{I}$ and target $(\langle (s', d') \rangle, q')$ realised by $\tau_{1,j}$ is represented by the family of transitions $(\vec{a}, (q, \sigma, i, \vec{s})) \rightarrow (\sigma(j), d', (q', \tau_{1,j} \circ \sigma, P \cup \{q\}, i, \vec{s}))$ for $\vec{a}$ such that $a_{\sigma(j)} = s'$.
- Each edge e in M , of source $(\langle (s, d) \rangle, q)$ and target $(\langle (s', d') \rangle, q')$ realised by $\tau_{1,j}$ is represented by the family of transitions $(\vec{a}, (q, \sigma, i, \vec{s})) \rightarrow (\sigma(j), d', (q', \tau_{1,j} \circ \sigma, i, \vec{s}))$ for all $\vec{a}$ such that $a_{\sigma(j)} = s'$ and $a_{\sigma(1)} = s$.
- Each edge e in M , of source $(\langle (s, d) \rangle, q)$ and target $(\langle x \rangle, q)$ with $q \in \mathcal{J}$ realised by $\tau_{1,j}$ is represented by:
 - the family of transitions $(\vec{a}, (q, \sigma, i, \vec{s})) \rightarrow (\sigma(j), d', (q', \tau_{1,j} \circ \sigma, i, \vec{s}))$ for $\vec{a}$ such that $a_{\sigma(1)} = s$ and $a_{\sigma(j)} = s'$ and $q \neq i$;
 - the family of transitions $(\vec{a}, (q, \sigma, i, \vec{s})) \rightarrow \text{reject}$ for $\vec{a} = \vec{s}$ and $i = q$;
- For each $i \in \mathcal{J}$ and $\vec{s} \in \{\star, 0, 1\}^N$, there is a transition $(\vec{a}, \text{init}) \rightarrow (\vec{a}, (i, \text{Id}, i, \vec{a}))$.

► **Definition 41.** For all integer N and Γ_N -essential m_N -machine G , we denote $[G]$ the N -head automaton described above.

The reader can convince herself it is a consequence of the definition of $\{M\}$ that, given a word w as input, it follows nondeterministically all alternating paths between $\tilde{G}$ and $\tilde{W}_w \otimes \text{Id}_{\tilde{r}}$ where $\tilde{r} = \{x\} \times \{\times_{i=1}^N [k_i/k, (k_i + 1)/k] \mid 0 \leq k_i \leq k - 1\}$. From this fact and the fact that such a cycle has to go through one of the vertices in $\tilde{r}$, we obtain the following proposition.

► **Proposition 42.** Let G be a Γ_N -essential m_N -machine, w a word and $!W$ be a word representation of w . There is an alternating cycle between $G :: !W$ and $\text{Id}_{\langle x \rangle}$ going through $\langle x \rangle$ if and only if the automaton $[G]$ rejects when given w as input.

► **Theorem 43.** Any language computed by a m_N -machine w.r.t. $\mathcal{T}_-$ is computed by a deterministic N -head automaton.

Proof. The proof consists in combining previous statements. Let G be a m_N -machine. Then there exists a Γ_N -essential m_N -machine H such that $\mathcal{L}^{\mathcal{T}_-}(G) = \mathcal{L}^{\mathcal{T}_-}(H)$. Now, we have defined the automaton $\{H\}$ which, by Proposition 42, rejects an input w if and only if there is an alternating path between H and $\tilde{W}_w \otimes \text{Id}_{\langle x \rangle}$ going through $\langle x \rangle$. But this is equivalent, by Proposition 40, to the fact that $H :: !W_w$ is not orthogonal to $\mathcal{T}_-$. Summing up, we have shown that $\{M\}$ rejects w if and only if $w \notin \mathcal{L}^{\mathcal{T}_-}(G)$. ◀

5 Conclusion and Perspectives

Combining Theorem 36 and Theorem 43, we obtain the characterisation of the hierarchy of sublinear complexity classes announced in the introduction.

► **Theorem 44.** For all $i \in \mathbf{N}^* \cup \{\infty\}$, $\mathbf{Pred}(m_i) = \text{CO2ANFA}(i)$

In particular, the microcosm m_1 characterises the class of regular languages, while the microcosm m_∞ characterises the class CONLOGSPACE .

Future work includes the extension of the techniques to other complexity classes. A similar characterisation of the class of polynomial time predicates should be easily obtained following the recent result by the author and coauthors [2]. This should lead to PTIME and not CONPTIME since the characterisation is based on pushdown automata [7]. Following the syntactic characterisation obtained by Baillot [5] by interpreting (some) Turing machines

as ELL proofs, one can expect a characterisation of the nondeterministic polynomial time class CONP TIME. As explained in an overview and perspective paper [26], the results will be adapted for deterministic and probabilistic classes.

The Logical View. As explained above, the set $\mathbf{Pred}(m_i)$ can be understood both as semantic restrictions over the set of computable predicates in the model of Elementary Linear Logic described by the microcosm p (Theorem 14), or as the set of computable predicates in a model of a modified linear logic lying in between MALL and ELL. Future work in this direction includes the understanding of these intermediate logics, and how they can be described syntactically. Let us provide here a first intuition in this regard. One should notice that functorial promotion is implemented by two steps: the first step uses permutations to prevent the interaction of the information encoded in $[0, 1]$ during exponentiation; a second step then takes the two copies of $[0, 1]$ and encodes them into a single one by using the function $[\cdot, \cdot]$, obtaining a graphing in the image of the exponentiation operation. The microcosms considered here are obtained by removing the latter function, hence preventing this second step. As a consequence, the models allow for limited composition of exponentiated maps: each new composition requires the use of a new copy of $[0, 1]$, and disallow to view those as exponentiated objects themselves. As a consequence, the intuition is that the characterisation of CONLOGSPACE obtained above corresponds to a restriction of linear logic where arbitrary compositions of exponentiated objects is possible but the resulting object cannot be seen as an exponential object. In some manner, the corresponding system should allow for *external* functorial promotion, in the same sense that countable models of set theory allow for *external* bijections between any two sets regardless of their cardinality in the model.

The Geometric View. As explained in the introduction and not developed yet, the results we obtain are of a geometrical nature. Indeed, the class are here characterised by microcosms which are (in this case) actions of groups on a measured space. Indeed, the microcosm m_i is obtained from the set of translations on $\mathbf{Z}$ together with the set of maps induced by the action of the set $\mathfrak{S}_i$ of permutations over $\{1, \dots, i\}$ onto the space $[0, 1]^i$. One should notice that the translations will always exist in any other characterisation of complexity classes using the techniques explained in this paper: this is because they are needed to interact with the input. Therefore, only the action of the group $\mathfrak{S}_i$ is of importance here. Future work will therefore consider how these group actions are related to the characterisations. Since the integer representation is independent from the group action, it is not difficult to convince oneself that, on one hand, any equivalent – homotopic – transformation of the space will give rise to the same complexity class. On the other hand, the group actions considered in this paper can be shown to be non-homotopic by using mathematical invariants [9]. Together with the separation result (Theorem 28), this lead the author to the conjecture that the converse holds [26], i.e. that non-equivalent group actions yield distinct complexity classes.

References

- 1 Clément Aubert, Marc Bagnol, Paolo Pistone, and Thomas Seiller. Logic programming and logarithmic space. In Garrigue [10], pages 39–57. doi:10.1007/978-3-319-12736-1_3.
- 2 Clément Aubert, Marc Bagnol, and Thomas Seiller. Unary resolution: Characterizing ptime. In *FOSSACS 2016*, 2016.
- 3 Clément Aubert and Thomas Seiller. Characterizing co-nl by a group action. *Mathematical Structures in Computer Science*, 2016. doi:10.1017/S0960129514000267.
- 4 Clément Aubert and Thomas Seiller. Logarithmic space and permutations. *Information and Computation*, 2016. doi:10.1016/j.ic.2014.01.018.

- 5 Patrick Baillot. Elementary linear logic revisited for polynomial time and an exponential time hierarchy. In Hongseok Yang, editor, *APLAS*, volume 7078 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2011.
- 6 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2, 1992.
- 7 Stephen A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 18(1):4–18, January 1971. URL: <http://doi.acm.org/10.1145/321623.321625>, doi:10.1145/321623.321625.
- 8 Vincent Danos and Jean-Baptiste Joinet. Linear logic & elementary time. *Information and Computation*, 183(1):123–137, 2003. doi:10.1016/S0890-5401(03)00010-5.
- 9 Damien Gaboriau. Coût des relations d'équivalence et des groupes. *Inventiones Mathematicae*, 139:41–98, 2000.
- 10 Jacques Garrigue, editor. *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, volume 8858 of *Lecture Notes in Computer Science*. Springer, 2014. doi:10.1007/978-3-319-12736-1.
- 11 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987. doi:10.1016/0304-3975(87)90045-4.
- 12 Jean-Yves Girard. Geometry of interaction V: Logic in the hyperfinite factor. *Theoretical Computer Science*, 412:1860–1883, 2011.
- 13 Jean-Yves Girard. Normativity in logic. In Peter Dybjer, Sten Lindström, Erik Palmgren, and Göran Sundholm, editors, *Epistemology versus Ontology*, volume 27 of *Logic, Epistemology, and the Unity of Science*, pages 243–263. Springer, 2012.
- 14 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, April 1992. doi:10.1016/0304-3975(92)90386-T.
- 15 Martin Hyland and Andrea Schalk. Glueing and orthogonality for models of linear logic. *Theoretical Computer Science*, 294, 2003. doi:10.1016/S0304-3975(01)00241-9.
- 16 D. Leivant and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundam. Inform.*, 19, 1993.
- 17 Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: Substitution and poly-space. *Lecture Notes in Computer Science*, 933, 1994.
- 18 Buckhard Monien. Transformational methods and their application to complexity problems. *Acta Informatica*, 6:95–108, 1976.
- 19 A. A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55, 1997.
- 20 Thomas Seiller. Interaction graphs: Multiplicatives. *Annals of Pure and Applied Logic*, 163:1808–1837, December 2012. doi:10.1016/j.apal.2012.04.005.
- 21 Thomas Seiller. *Logique dans le facteur hyperfini : géométrie de l'interaction et complexité*. PhD thesis, Université Aix-Marseille, 2012. URL: <http://tel.archives-ouvertes.fr/tel-00768403/>.
- 22 Thomas Seiller. Interaction graphs: Additives. *Annals of Pure and Applied Logic*, 167:95 – 154, 2016. doi:10.1016/j.apal.2015.10.001.
- 23 Thomas Seiller. Interaction graphs: Exponentials. *Logical Methods in Computer Science*, 2016. Under revision. URL: <http://arxiv.org/pdf/1312.1094>.
- 24 Thomas Seiller. Interaction graphs: Full linear logic. Submitted, 2016. URL: <http://arxiv.org/pdf/1504.04152>.
- 25 Thomas Seiller. Interaction graphs: Graphings. *Annals of Pure and Applied Logic*, 2016. Under revision. URL: <http://arxiv.org/pdf/1405.6331>.
- 26 Thomas Seiller. Towards a *Complexity-through-Realizability* theory. Submitted, 2016. URL: <http://arxiv.org/pdf/1502.01257>.