Interaction Graphs: Non-Deterministic Automata

THOMAS SEILLER, CNRS, France and University of Copenhagen, Denmark

This article 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.

CCS Concepts: • **Theory of computation → Complexity classes**; **Complexity theory and logic**; **Linear logic**; **Denotational semantics**; *Regular languages*; *Proof theory*;

Additional Key Words and Phrases: Implicit computational complexity, dynamic semantics, interaction graphs, automata, linear logic, measurable dynamics

ACM Reference format:
Thomas Seiller. 2018. Interaction Graphs: Non-Deterministic Automata. *ACM Trans. Comput. Logic* 19, 3, Article 21 (August 2018), 24 pages.
https://doi.org/10.1145/3226594

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 that 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 [6, 20] 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 the attempts to provide alternative characterisations, 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's landmark paper [7] 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 that 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

Thomas Seiller was supported by the European Commision's Marie Sklodowska-Curie Individual Fellowship (H2020-MSCA-IF-2014) 659920-ReACT.
Author's current address: T. Seiller, LIPN – UMR 7030 CNRS & Universtié Paris Nord, Institut Galilée – Université Paris 13, 99, Avenue Jean-Baptiste Clément, 93430 Villetaneuse, France.

https://doi.org/10.1145/3226594

then yields variants of linear logic having computational interest: this is how constrained linear logic systems, for instance, BLL [14] and ELL [9], 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 motivated by his work on Geometry of Interaction (GOI) models, and more precisely the hyperfinite GOI model [12]. Together with C. Aubert, the author showed how Girard's proposal leads to a characterisation of coNLOGSPACE [3, 22] 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 that 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 has its own interests, it is the author's belief that one should not forget the mathematical structure from which these characterisations originated. This sentiment is strengthened by the author's discovery of a correspondence between fragments of linear logic and a classification of maximal abelian subalgebras (MASAS) of von Neumann algebras [29]. The approach taken in this article is therefore quite orthogonal to the recent evolutions of the subject, as it aims at the obtention of a deeper understanding of how complexity classes can be related to the mathematics behind GOI models to provide complexity theorists with new techniques and invariants [24].

Contributions and Outline

The present work achieves three distinct goals related to the logic-based characterisations of complexity classes. First, 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 [1–4] and the actual semantics provided by GOI models. Second, we obtain characterisations of several classes that were not available using previous techniques. This is due to a change of perspective that allows new proof techniques, sensitive to more subtle differences. Third, 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 article is constructed as follows. Section 2 introduces the technical material about *interaction graphs models of linear logic*. This will allow us to define, in Section 3, the ambient model that will be used to obtain the characterisations. We also define the representation of binary words and the notion of $\mathfrak{m}_k$-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 $\mathfrak{m}_k$-machine coincide. Last, we discuss this result in Section 4, 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 [21, 25–28]. 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 represent 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.

For technical reasons explained in earlier papers [28], not all measurable maps can be used to represent edges. To be able to define models of linear logic, one has to restrict to non-singular measurable-preserving maps. We recall that a map f is a *non-singular transformation* from a measure space $\mathbf{X} = (X, \mathcal{B}, \mu)$ onto itself, if it is a measurable map such that $\mu(f(A)) = 0$ if and only if $\mu(A) = 0$. We say f is *measurable-preserving* when $f(A) \in \mathcal{B}$ whenever $A \in \mathcal{B}$.

Definition 2.1 (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 transformations $\mathbf{X} \to \mathbf{X}$, which has the structure of a monoid w.r.t. the composition of maps. That is, it is closed under function composition and contains the identity.

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.

Examples 1. For all examples considered in this section, we will restrict to the underlying measure space the real line $\mathbf{R}$ endowed with the Lebesgue measure. We first define the microcosm $\mathfrak{z}$ as the set of all integral translations on $\mathbf{R}$; i.e.,

$$\mathfrak{z} = \{T_k : \mathbf{R} \to \mathbf{R}, x \mapsto x + k \mid k \in \mathbf{Z}\}.$$

Notice that this microcosm is generated by the set $\{T_1, T_{-1}\}$.

Now, we can also define the microcosm $\mathfrak{h}$ of integral homotheties on $\mathbf{R}$; i.e.,

$$\mathfrak{h} = \{H_z : \mathbf{R} \to \mathbf{R}, x \mapsto z.x \mid z \in \mathbf{Z}\}.$$

For this microcosm, no finite generating set exists. The following (infinite) set is, however, generating: $\{H_p \mid (-p)$ is prime or equal to $1\}$.

These two microcosms are almost disjoint, as only the identity map on $\mathbf{R}$ belongs to both of them. They are, however, submonoids of several common microcosms; in particular, there exists a minimal such microcosm, namely, the monoid of all integral affine transformations; i.e.,

$$\mathfrak{aff} = \{A_{k,h} : \mathbf{R} \to \mathbf{R}, x \mapsto h.x + k \mid k, h \in \mathbf{Z}\}.$$

Finally, all microcosms on a measure space $\mathbf{X}$ are submonoids of *the largest microcosm on* $\mathbf{X}$ – called the *macrocosm* – defined as the set of all non-singular measurable-preserving transformations on $\mathbf{X}$.

We must point out that a more general notion of microcosm was introduced in a recent work by the author [26]; the restricted notion defined here is, however, easier to grasp and sufficient for our purposes in this article. We now define the notion of graphing.

Definition 2.2 (Graphing representative). We fix a measure space $\mathbf{X}$, a microcosm $\mathfrak{m}$, a monoid Ω, a measurable subset V^G of $\mathbf{X}$, and a finite set D^G. A (Ω-weighted) $\mathfrak{m}$-*graphing representative* G of *support* V^G and *dialect* D^G is a countable set,

$$\{(S_e^G, \mathrm{i}_\mathrm{e}^\mathrm{G}, \mathrm{o}_\mathrm{e}^\mathrm{G}, \phi_\mathrm{e}^\mathrm{G}, \omega_\mathrm{e}^\mathrm{G}) \mid \mathrm{e} \in \mathrm{E}^\mathrm{G}\},$$

where S_e^G is a measurable subset[1] of $V^G \times D^G$, ϕ_e^G is an element of $\mathfrak{m}$ such that $\phi_e^G(S_e^G) \subseteq V^G$, $\mathrm{i}_\mathrm{e}^\mathrm{G}, \mathrm{o}_\mathrm{e}^\mathrm{G}$ are elements of D^G, and $\omega_e^G \in \Omega$ is a *weight*. We will refer to the indexing set E^G as the set

[1]As D^G is considered as a discrete measure space, a measurable subset of the product is simply a finite collection of measurable subset indexed by elements of D^G.

of *edges*. For each edge $e \in E^G$ the set $S^G_e \times \{\mathrm{i}^{\mathrm{G}}_{\mathrm{e}}\}$ is called the source of e, and we define the *target* of e as the set $T^G_e \times \{\mathrm{o}^{\mathrm{G}}_{\mathrm{e}}\}$ where $T^G_e = \phi^G_e(S^G_e)$.

To provide some intuitions, we first ignore the dialect D^G, or equivalently, we consider D^G to be a singleton. Given an edge $e \in E^G$, the intuition is that the triple $(S^G_e, \phi^G_e, \omega^G_e)$ corresponds to the following information: the source S^G_e of the edge, the target $T^G_e := \phi^G_e(S^G_e)$ of the edge, the weight ω^G_e of the edge. Consequently, a graphing may be mapped to a graph whose edges are measurable subsets of $\mathbf{X}$. However, two different graphings may give rise to the same graph, as this mapping forgets about *how* the source is mapped to the target, i.e., which measurable map in the microcosm realises the edge.

The additional information of the elements $\mathrm{i}^{\mathrm{G}}_{\mathrm{e}}, \mathrm{o}^{\mathrm{G}}_{\mathrm{e}}$ corresponds to *control states*. Indeed, thinking of the finite set D^G as a set of control states is a good intuition that can be followed through this article. Building on this, one can define a weighted automaton from a graphing as follows: the automaton works on the (infinite) input alphabet consisting of all measurable subsets of $\mathbf{X}$ and has as set of states D^G; then each edge e defines a transition from S^G_e in state $\mathrm{i}^{\mathrm{G}}_{\mathrm{e}}$ to T^G_e in state $\mathrm{o}^{\mathrm{G}}_{\mathrm{e}}$. This mapping, however, is again non-injective as it does not account for *how* the source is mapped to the target.

Examples 2. We first consider an example of *deterministic graphing representative*, i.e., one such that every $x \in \mathbf{X}$ belongs to the source of at most one edge (up to a null measure set). For the sake of simplicity, the graphing representatives F, G we consider are such that $D^F = D^G = \{\star\}$, i.e., they have a unique control state, and all weights will be equal to 1; they are then defined by $V^F = V^G = [0, 2[$ and

$$F = \{([0, 1[, \star, \star, x \mapsto x + 1, 1), ([1, 2[, \star, \star, x \mapsto x - 1, 1)\},$$

$$G = \{([0, 1[, \star, \star, x \mapsto x + 1, 1), ([1, 2[, \star, \star, x \mapsto 2 - x, 1)\}.$$

Note that these two examples give rise to the same graph and the same automata through the mapping just explained above. They are, however, quite different. In particular, using the notations of Examples 1, the graphing F is a $\mathfrak{t}$-graphing while G is not. Indeed, G is neither a $\mathfrak{t}$-graphing or a $\mathfrak{h}$-graphing; it is, however, a $\mathfrak{aff}$-graphing.

Even though the intuitions given above are good to keep in mind, they are only approximations of the actual notion of graphing. Indeed, a graphing is defined as an equivalence class of graphing representatives. In particular, a graphing is not a specific set of edges realised by elements of a given microcosms: it is the generalised measurable dynamical system underlying a specific representation. In particular, both intuitions of graphings as graphs and automata fail to convey this idea that we now illustrate on an example.

Examples 3. We consider the graphing representative F defined in Examples 2. We define the graphing representative H defined by $D^H = \{\star\}$, $V^H = [0, 2[$, and

$$H = \{(]0, 1/2[, \star, \star, x \mapsto x + 1, 1), (]1/2, 1[, \star, \star, x \mapsto x + 1, 1), ([1, 2[, \star, \star, x \mapsto x - 1, 1)\}.$$

The notion of graphing should be such that F and H are representative of the same graphing. To understand this, consider the graphing H' defined by $D^{H'} = \{\star\}$, $V^{H'} = [0, 2[$, and

$$H' = \{([0, 1/2[, \star, \star, x \mapsto x + 1, 1), ([1/2, 1[, \star, \star, x \mapsto x + 1, 1), ([1, 2[, \star, \star, x \mapsto x - 1, 1)\}.$$

Then, H' is a *refinement* of H in that we only replaced the edge $([0, 1[, \star, \star, x \mapsto x + 1, 1)$ by the two edges $([0, 1/2[, \star, \star, x \mapsto x + 1, 1)$ and $([1/2, 1[, \star, \star, x \mapsto x + 1, 1)$ to define H' from F. Moreover, H' is *almost-everywhere equal* to H.

As illustrated by the example, it is natural to identify graphing representatives w.r.t. almost-everywhere equality and a notion of *refinement*, both combined in the following formal definition, which is studied in earlier work [28].

Definition 2.3. A graphing representative F is a refinement of a graphing representative G if there exists a partition[2] $(E^F_e)_{e\in E^G}$ of E^F, such that

$$\forall e \in E^G, \cup_{f\in E^F_e} S^F_f =_{a.e.} S^G_e; \qquad \forall e \in E^G, \forall f \neq f' \in E^F_e, \mu(S^F_f \cap S^F_{f'}) = 0;$$

$$\forall e \in E^G, \forall f \in E^F_e, \omega^F_f = \omega^G_e; \qquad \forall e \in E^G, \forall f \in E^F_e, \phi^F_f = \phi^G_e.$$

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 [28], 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.

2.2 Paths and Execution

In previous work, the author showed how to build denotational models of *types*, or *formulas*, by using graphings (over a space $\mathbf{X}$ chosen once and for all) to interpret *programs*, or *proofs*, depending on which side of the proofs-as-programs correspondence we are standing on. These denotational models should be described as *dynamic*, as they represent program execution, or the cut-elimination procedure, as a non-trivial operation in the semantics. In that aspect, they are distinguished from so-called *static* denotational models in which a proof and its normal form have the same "denotation." In the specific models built from graphings, the dynamic aspect is represented by the operation of *execution*, based on the computation of alternating paths.

An *alternating* path between two $\mathfrak{m}$-graphings F, G is a sequence of edges $\pi = e_1, e_2, \ldots, e_k$ satisfying the following two conditions:

- e_i in E^G if and only if $e_{i+1} \in E^F$, and $\mathtt{o}_{\mathtt{e}_i} = \mathtt{i}_{\mathtt{e}_{i+2}}$;
- every measurable set $(\phi_{e_i} \circ \phi_{e_{i-1}} \circ \ldots \phi_{e_1})(S_{e_1})$ is of strictly positive measure.[3]

We denote $\mathrm{AltPath}(F, G)$ the set of such paths. A given path naturally represents the composition $\phi_\pi = \phi_{e_k} \circ \cdots \circ \phi_{e_1}$, which belongs to $\mathfrak{m}$, since the latter is a monoid. We define the source of π as $S_\pi \times \{\mathtt{i}_{\mathtt{e}_1}, \mathtt{i}_{\mathtt{e}_2}\}$, where S_π is defined as the set of all x such that for all i, $\phi_{e_i} \circ \cdots \circ \phi_{e_1}(x) \in S_{e_{i+1}}$. 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 $[\pi]^o_o(C)$ as the path representing the same map as π, and whose source has been restricted to $S^{\forall C} \times \{\mathtt{i}_{\mathtt{e}_1}, \mathtt{i}_{\mathtt{e}_2}\}$, with $S^{\forall C}_\pi = (S_\pi \cap \bar{C} \cap (\phi_\pi)^{-1}(\bar{C}))$, 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. The execution between graphings F, G of respective supports $V + C$ and $W + C$ is then defined as the graphing $F :: G$ of support $V + W$ consisting of all $[\pi]^o_o(C)$ for π an alternating path between F and G.

Definition 2.4 (Execution). Let F and G be graphings such that $V^F = V \uplus C$ and $V^G = C \uplus W$ with $V \cap W$ of null measure. Their *execution* $F :: G$ is the graphing of support $V \uplus W$ and dialect

[2] We allow the sets E^F_e to be empty.

[3] In particular, $S_{e_{i+1}} \cap T_{e_i}$ is non-negligible, i.e., of strictly positive measure.

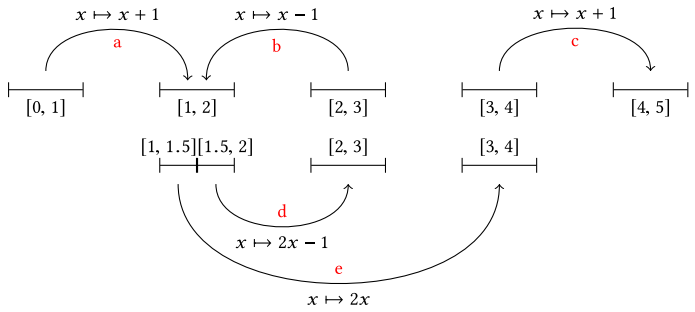


Fig. 1. Two graphings F (above) and G (below).

$D^F \times D^G$ defined as the set of all $[\pi]_o^o(C)$, where π is an alternating path between F and G:

$$F :: G = \{(S_\pi^{\forall C}, (\mathrm{i}_{\mathrm{e}_1}, \mathrm{i}_{\mathrm{e}_2}), (\mathrm{o}_{\mathrm{e}_{n-1}}, \mathrm{o}_{\mathrm{e}_n}), \phi_\pi, \omega_\pi) | \pi = \mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_n \in \mathrm{AltPath(F, G)}\}.$$

Examples 4. Consider the two graphings F and G shown in Figure 1 (F is shown at the top of the figure; G at the bottom). Their execution is then the graphing with the following countably infinite family of paths $\{a(db)^k ec\}_{k=0}^{\infty}$, where $a(db)^k ec$ is of source $[(2^{k-1}-1)/2^{k-1}, (2^k-1)/2^k]$ and realised by the function $x \mapsto 2^k x - 2^k + 6$.

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-gluing 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 [28], it will be enough for our purpose to consider:

- its restriction to measure-preserving maps;
- a fixed parameter map $m : \Omega \to \mathbf{R}_{\geqslant 0} \cup \{\infty\}$.

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 $\mathrm{Rep}(F, G)$ of cycles $\pi = (e_i)_{i=0}^{n-1}$ such that (1) $\pi^k \in \mathrm{Rep}(F, G)$ (k a non-zero integer) implies $\pi \in \mathrm{Rep}(F, G)$; (2) $\pi \in \mathrm{Rep}(F, G)$ implies $(e_{i+k})_{i=0}^{n-1} \notin \mathrm{Rep}(F, G)$ (where $i + k$ is computed in $\mathbf{Z}/n\mathbf{Z}$). More technical details about the definition (in particular, a definition in the general case) and a proof that the considered measurement is independent from this choice of representative is found in previous work by the author [28].

For the purpose of this work, let us simply point out the relationship between the orthogonality deduced from the measurement and the *correctness criterion* for proof nets. The interested reader can consult earlier work by the Naibo, Petrolo, and the author [19], where it is explained how the correctness criterion can lead to a notion of *type* when understood as an orthogonality. The orthogonality considered here therefore generalises the correctness criterion in a way that allows for the use of weights. This is in particular useful in this article as weights are used to distinguish

between forbidden cycles (i.e., proofs should be *acyclic* following the correctness criterion) and cycles that may be part of an automaton and should therefore be allowed.[4]

Definition 2.5. The measurement between two graphings (realised by measure-preserving maps) is defined as

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

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

We now describe the models. For technical reasons explained in previous papers [21, 25], 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:

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

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

Definition 2.6. 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 2.7. Two projects (a, A) and (b, B) are orthogonal – written $(a, A) \perp (b, B)$ – when they have equal support and $[\![(a, A), (b, B)]\!] \neq 0, \infty$. We also define the orthogonal of a set E as $E^\perp = \{(b, B) : \forall (a, A) \in A, (a, A) \perp (b, B)\}$ and write $E^{\perp\perp}$ the double-orthogonal $(E^\perp)^\perp$.

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

Definition 2.8. A *conduct* of support V^A is a set $\mathbf{A}$ of projects of support V^A such that $\mathbf{A} = \mathbf{A}^{\perp\perp}$. A *behaviour* is a conduct such that whenever (a, A) belongs to $\mathbf{A}$ (respectively, $\mathbf{A}^\perp$) and for all $\lambda \in \mathbf{R}$, then $(a, A + \lambda\emptyset)$ belongs to $\mathbf{A}$ (respectively, $\mathbf{A}^\perp$) as well. If both $\mathbf{A}$ and $\mathbf{A}^\perp$ are non-empty, then we say $\mathbf{A}$ is *proper*.

Conducts provide a model of Multiplicative Linear Logic. The connectives $\otimes$, $\multimap$ are defined as follows: if $\mathbf{A}$ and $\mathbf{B}$ are conducts of disjoint supports V^A, V^B, i.e., $V^A \cap V^B$ is of null measure, then:

$$\mathbf{A} \otimes \mathbf{B} = \{\mathfrak{a} :: \mathfrak{b} \mid \mathfrak{a} \in \mathbf{A}, \mathfrak{b} \in \mathbf{B}\}^{\perp\perp},$$
$$\mathbf{A} \multimap \mathbf{B} = \{\mathfrak{f} \mid \forall \mathfrak{a} \in \mathbf{A}, \mathfrak{f} :: \mathfrak{b} \in \mathbf{B}\}.$$

However, to define additive connectives, one has to restrict the model to behaviours. In this article, 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 2.9 ([25, Proposition 60]). *If* $\mathbf{A}$ *is a proper behaviour, then* $(a, A) \in \mathbf{A}$ *implies* $a = 0$.

[4] In this respect, the current work differs from the characterisation of coNLogspace obtained by Aubert and the author [3], since in this earlier paper the authors had to restrict to *acyclic* automata.

Finally, let us mention the fundamental theorem for the interaction graphs construction in the restricted case we just exposed.[5]

Theorem 2.10 ([28, 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 that interpret second-order quantification [28] as well as exponential connectives [26, 27].

3 INTEGERS, MACHINES, TESTS

We will now define a specific model that will be studied throughout the rest of the article. After defining the underlying measure space, we will define a family of microcosms. The largest of those microcosms, namely $\mathfrak{p}$, will be used to define the model with which we will work – the surrounding universe. We will start by showing that this is a model Elementary Linear Logic (ELL), a logic fragment expressive enough to define a representation of binary words. The smaller microcosms $\mathfrak{m}_i$ will be used to define submodels of this surrounding universe, which will characterise small complexity classes.

Since the resulting model is of Elementary Linear Logic (ELL), one can represent binary words using the type of binary lists in ELL. The corresponding proofs can then be interpreted as graphings (or rather as projects $(0, G)$ with G a graphing), but a single proof can be interpreted as a myriad of graphings depending on choice in the interpretation's definition. Consequently, an ELL proof representing a binary word will be interpretable by many different graphings. Those graphings, however, are all obtained as representations of the same graph, corresponding to the set of axiom rules in the corresponding proof net. We refer the reader to an earlier paper for an illustrated discussion of how binary words can be represented as graphs [3]; we define in the next section the interpretation of binary words directly.

Once the type of binary words is defined, one can consider the type of binary predicates in the model. Among those graphings realising this type, we consider only the *finite* ones, i.e., those that can be described by a finite number of edges. These objects are called *machines*, and can be further classified according to the monoids of measurable maps used to realise their edges. This leads to a notion of $\mathfrak{m}$-machine for a microcosm $\mathfrak{m}$, which is a submonoid of $\mathfrak{p}$. In a way, we are therefore defining subsets of the type of predicates in a model of ELL. However, let us recall that each such submonoid $\mathfrak{m}$ describes a model of MALL (at least); consequently another reading of this is to understand $\mathfrak{m}$-machines as finite graphings in the type of predicates of a smaller model described by $\mathfrak{m}$. In particular, these models are not complete w.r.t. MALL and should satisfy additional axioms. Since these models characterise small complexity classes, one could try to derive from these models logical systems describing (space) sub-linear complexity classes.

3.1 General Situation

Notice that while previous work (and the previous section) 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 $a \cdot \mathbf{1}$. On this set of weights, we will consider the fixed parameter map $m(x, y) = xy$ in the following.

[5] The general construction allows for other sets of weights as well as whole families of measurements [28].

In practice, most graphings considered in this article do not use weights different from 1 (i.e., $(1,0)$), except for the *tests* (Definition 3.12). We will therefore allow ourselves to define graphing representatives without mentioning the weights, implying that those are all equal to 1.

Moreover, graphings were 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* [27], i.e., indistinguishable in the model. Formally, this is expressed as the fact that for every graphing H, the measurement $[\![F,H]\!]_m$ coincides with the measurement $[\![G,H]\!]_m$. Consequently, we will always consider in the following that dialects are chosen as initial segments of the natural numbers, i.e., sets $[n]=\{0,1,\dots,n\}$.

Definition 3.1 (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 [26], 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 $(a,b)\cdot\mathbf{s}$ for the sequences whose first two elements are a,b (and we abusively write $a\cdot\mathbf{s}$ instead of $(a)\cdot\mathbf{s}$). 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 3.2 (Microcosms). For all integer $i\geqslant 1$, we consider the microcosm $\mathfrak{m}_i$ generated by the translations $\mathtt{t}_z:(\mathrm{x},\mathbf{s})\mapsto(\mathrm{x}+\mathrm{z},\mathbf{s})$ for all integer z, and the permutations $\mathtt{p}_\sigma:(\mathrm{x},\mathbf{s})\mapsto(\mathrm{x},\sigma(\mathbf{s}))$ for all permutation σ such that $\sigma(k)=k$ for all $k>i$. We write $\mathfrak{m}_\infty$ the union $\cup_{i>1}\mathfrak{m}_i$.

Let us denote by $a\bar{+}b$ the fractional part of the sum $a+b$. We also define the microcosms $\bar{\mathfrak{m}}_i$ as the smallest microcosm containing $\mathfrak{m}_i$ and all translations $\mathtt{t}_\lambda:(\mathrm{x},\mathrm{a}\cdot\mathbf{s})\mapsto(\mathrm{x},(\mathrm{a}\bar{+}\lambda)\cdot\mathbf{s})$ for λ in $[0,1]$.

We now define a bijective measure-preserving pairing function: $[\cdot,\cdot]:[0,1]^2\to[0,1]$. 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

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

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

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

Definition 3.3. Given a graphing $G=\{(S_e^G,\mathrm{i}_\mathrm{e}^\mathrm{G},\mathrm{o}_\mathrm{e}^\mathrm{G},\phi_\mathrm{e}^\mathrm{G})$ of dialect $D=[n]$, we define the promotion $!G$ of G as the following graphing of dialect $[0]$:

$$\{(\mathsf{push}^n_{\mathrm{i}_\mathrm{e}^\mathrm{G}}(S_e^G),0,0,\mathsf{push}^n_{\mathrm{i}_\mathrm{e}^\mathrm{G},\mathrm{o}_\mathrm{e}^\mathrm{G}}(\phi_e^G))\mid e\in E^G\}.$$

This previous definition is a *perennisation*, as defined in earlier papers [22, 27]; i.e., it maps arbitrary graphings to graphings with trivial dialect [0]. This is to ensure that all graphings of the form $!A$ are *duplicable*: since one can always find a graphing C such that $C::A\simeq A\otimes A$ for all A with a trivial dialect [27, Proposition 36], we can implement *contraction* on graphings of the form $!A$, and by extension on conducts generated by graphings of this form.

Definition 3.4. Given a behaviour $\mathbf{A}$, we define the conduct $!\mathbf{A}$ as the set $\{(0,!G)\mid G\in\mathbf{A}\}^{\perp\perp}$.

Following the remark above, given any conduct $\mathbf{A}$ one can always define a graphing C implementing contraction, i.e., such that $(0,C)\in !\mathbf{A}\multimap !\mathbf{A}\otimes!\mathbf{A}$.

Remark 1. 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 [27, Corollary 57].

Theorem 3.5. *Consider the microcosm* $\mathfrak{p}$ *generated by* $\bar{\mathfrak{m}}_\infty$ *together with the additional maps* $\mathtt{pair}$ *and* $\mathtt{pair}^{-1}$*, where* $\mathtt{pair} : (a,(x,y)\cdot \mathbf{s}) \mapsto (a,[x,y]\cdot \mathbf{s})$*. For any microcosm containing* $\mathfrak{p}$*, the set of conducts and behaviours is a model of Elementary Linear Logic.*

Proof. We only need to check that functorial promotion can be implemented, as contraction is automatically satisfied [27] and the fact that it is a model of MALL follows from Theorem 2.10. The technique is similar as what is used in previous papers [26, 27]. First, we notice the maps $\mathtt{inl} = (a,[x,[y,z]]\cdot \mathbf{s}) \mapsto (a,[[x,y],z]\cdot \mathbf{s})$ and $\mathtt{inr} = (a,[x,[y,z]]\cdot \mathbf{s}) \mapsto (a,[[y,x],z]\cdot \mathbf{s})$ belong to the microcosm $\mathfrak{p}$. Then, given $F \in \mathbf{A} \multimap \mathbf{B}$ and $A \in \mathbf{A}$, we can check that $\mathtt{inl}(!F) :: \mathtt{inr}(!A)$ is equivalent to $!(F :: A)$, which is an element of $!\mathbf{B}$. □

3.2 Representation of Binary Words

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

Notations 1. We write $\Sigma^{\Updownarrow}$ *the set* $\Sigma_\star \times \{\mathtt{in},\mathtt{out}\}$*. We also denote by* $\Sigma^{\Updownarrow}_{\mathtt{a},\mathtt{r}}$ *the set* $\Sigma^{\Updownarrow} \cup \{\mathtt{a},\mathtt{r}\}$*, where* $\mathtt{a}$ *(respectively,* $\mathtt{r}$*) stand for* $\mathtt{accept}$ *(respectively,* $\mathtt{reject}$*).*

Notations 2. We fix once and for all an injection Ψ *from the set* $\Sigma^{\Updownarrow}_{\mathtt{a},\mathtt{r}}$ *to intervals in* $\mathbf{R}$ *of the form* $[k,k+1]$ *with* k *an integer. For all* $f \in \Sigma^{\Updownarrow}_{\mathtt{a},\mathtt{r}}$ *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* $Y = [0,1]^{\mathbf{N}}$*, then we omit the subscript and write* $\langle f\rangle$*. The notation extends to any subset* S *of* $\Sigma^{\Updownarrow}_{\mathtt{a},\mathtt{r}}$*, i.e.,* $\langle S\rangle$ *is the (disjoint) union* $\cup_{f\in S}\langle f\rangle$*.*

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

$$\begin{array}{lll} V^{\bar{W}_\mathtt{w}} = \Sigma^{\Updownarrow}, & s^{\bar{W}_\mathtt{w}} = & (r,i) \mapsto (a_i,\mathtt{out},\mathrm{i}) \\ D^{\bar{W}_\mathtt{w}} = [k], & & (l,i) \mapsto (a_i,\mathtt{in},\mathrm{i}), \\ E^{\bar{W}_\mathtt{w}} = \{r,l\}\times[k], & t^{\bar{W}_\mathtt{w}} = & (r,i) \mapsto (a_{i+1},\mathtt{in},\mathrm{i}+1 \bmod \mathrm{k}+1) \\ & & (l,i) \mapsto (a_{i-1},\mathtt{out},\mathrm{i}-1 \bmod \mathrm{k}+1). \end{array}$$

This graph is the discrete representation of $\mathtt{w}$. Detailed explanations on how these graphs relate to the proofs of the formula BList can be found in earlier work [3, 22].

Definition 3.6. Let $\mathtt{w}$ be a word $\mathtt{w} = \star a_1 a_2 \dots a_k$ over the alphabet Σ. We define the word graphing $W_\mathtt{w}$ of support $\langle\Sigma^{\Updownarrow}\rangle$ and dialect $D^{\bar{W}_\mathtt{w}}$ by the set of edges $E^{\bar{W}_\mathtt{w}}$ and for all edge e:

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

Notations 3. We write $\mathtt{Gp}(\mathtt{w})$ *the set of word graphings for* $\mathtt{w}$*. It is defined as the set of graphings obtained by renaming the dialect* $D^{\bar{W}_\mathtt{w}}$ *w.r.t. an injection* $[k] \to [n]$*.*

Definition 3.7. Given a word $\mathtt{w}$, a *representation of* $\mathtt{w}$ is a graphing $!L$ where L belongs to $\mathtt{Gp}(\mathtt{w})$. The set of representations of words in Σ is denoted $\sharp\mathbf{W}_2$, the set of representations of a specific word $\mathtt{w}$ is denoted $\mathbf{Rep}(\mathtt{w})$.

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

Definition 3.8. We define the (unproper) behaviour $\mathbf{NBool}$ as $\mathbf{T}_{\langle\mathtt{a},\mathtt{r}\rangle}$, where for all measurable sets V the behaviour $\mathbf{T}_V$ is defined as the set of all projects of support V. For all microcosms $\mathfrak{m}$, 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 article 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 $\mathfrak{m}$-machines – will be *realisations* of graphs as $\mathfrak{m}$-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 $\mathfrak{m}$-machines can be followed through the rest of this article.

Definition 3.9. We use the notation $G \leqslant H$ for "G is a refinement of H" for the notion of refinement explained in Section 2.1. A $\mathfrak{m}$-graphing G is *finite* when there exists a $\mathfrak{m}$-graphing H such that $G \leqslant H$ and the set of edges E^H is finite.

Definition 3.10. A *nondeterministic predicate* $\mathfrak{m}$*-machine* over the alphabet Σ is a finite $\mathfrak{m}$-graphing belonging to $\mathbf{Pred}(\mathfrak{m})$ 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 Section 2.2. The result of the execution is an element of **NBool**, i.e. in some ways a generalised Boolean value.[6]

Definition 3.11 (Computation). Let M be a $\mathfrak{m}$-machine, $\mathtt{w}$ a word over the alphabet Σ and $!L \in !\mathbf{Nat}_2$. The *computation* of $\mathfrak{M}$ 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 the consideration of 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 3.12 (Tests). A *test* is a family $\mathcal{T} = \{\mathtt{t}_i = (t_i, T_i) \mid i \in I\}$ of projects of support $\langle \mathtt{a}, \mathtt{r} \rangle$.

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

$$\mathcal{L}^{\mathcal{T}}_{\exists}(M) = \{\mathtt{w} \in \Sigma^* \mid \forall \mathtt{t}_i \in \mathcal{T}, \exists \mathfrak{w} \in \mathbf{Rep}(\mathtt{w}), M :: \mathfrak{w} \perp\!\!\!\perp \mathtt{t}_i\},$$
$$\mathcal{L}^{\mathcal{T}}_{\forall}(M) = \{\mathtt{w} \in \Sigma^* \mid \forall \mathtt{t}_i \in \mathcal{T}, \forall \mathfrak{w} \in \mathbf{Rep}(\mathtt{w}), M :: \mathfrak{w} \perp\!\!\!\perp \mathtt{t}_i\}.$$

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

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

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

[6] If one were working with "deterministic machines" [24], then 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 4.1. A *two-way multihead automaton* $\mathtt{M}$ with k heads is defined as a tuple $(\Sigma, Q, \to)$, where $\to \subseteq \left(\Sigma_\star^k \times Q\right) \times ((\{1, \ldots, k\} \times \{\mathtt{in}, \mathtt{out}\}) \times Q)$ is the *transition relation* of $\mathtt{M}$. The automaton $\mathtt{M}$ is *deterministic* when the relation $\to$ 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 $\cup_{k \geqslant 1} \mathbf{2nfa}(k)$ is denoted by $\mathbf{2nfa}$.

Definition 4.2. We denote co2Nfa(k) the set of languages accepted by automata in $\mathbf{2nfa}(k)$, where an automaton $\mathtt{M}$ accepts a word $\mathtt{w}$ if and only there are no computation traces of $\mathtt{M}$ given $\mathtt{w}$ as input leading to a rejecting state.

The set of languages Regular = 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 4.3 (Monien [18]). *For all k, the set* co2Nfa(k) *is a* strict *subset of* co2Nfa(k+1).

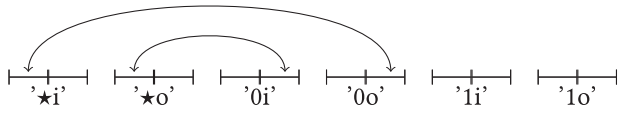
Theorem 4.4. $\cup_{k \geqslant 1}$co2Nfa(k) = coNLogspace.

We will now show how k-head multihead automata correspond to $\mathfrak{m}_k$-machines. Before going into the technical details, we show some examples of graphing representations of integers, machines, and computations taken from an unpublished overview and perspective paper by the author [24].

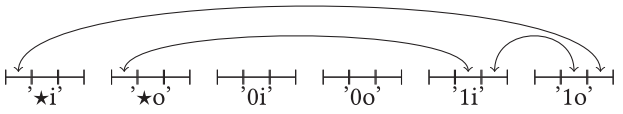
4.2 Examples

Examples of Words. We here work two examples to illustrate how the representation of computation by graphings works out. First, we give the representation as graphings of the lists $\star 0$ (Figure 2(a)), $\star 11$ (Figure 2(b)), and $\star 01$ (Figure 2(c)). In the illustrations, the vertices, e.g., "0i", "0o", represent disjoint segments of unit length, e.g., $[0, 1]$, $[1, 2]$. As mentioned in the caption, the plain edges are realised as translations. Notice that those edges are cut into pieces (two pieces each for the list $\star 0$, and three pieces each for the others). This is due to the fact that these graphings represent exponentials of the word representation: the exponential splits the unit segment into as many pieces as the length of the list; intuitively each piece of the unit segments correspond to the "address" of each bit. The edges then explicit the ordering: each symbol comes with pointers to its preceding and succeeding symbols. For example, in the case of the first "1" in the list $\star 11$, there is an edge from "1i" to "$\star$o" representing the fact that the preceding symbol was "$\star$", and there is an edge from "1o" to "1i" representing the fact that the next symbol is a "1"; notice, moreover, that these edges move between the corresponding addresses.

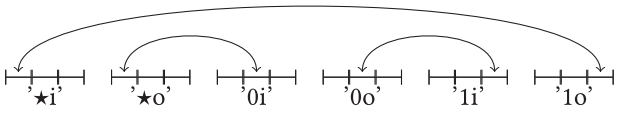
Examples of Machines. We will now study two examples of machines, their representation as graphings, and illustrate how computation is captured by the execution, i.e., the computation of a set of alternating paths. These examples are, however, incorrect w.r.t. the framework and translation shown in the next section. Indeed, the machines we consider do not satisfy the requirement



(a) The graphing representing the word ⋆0.



(b) The graphing representing the word ⋆11.



(c) The graphing representing the word ⋆01.

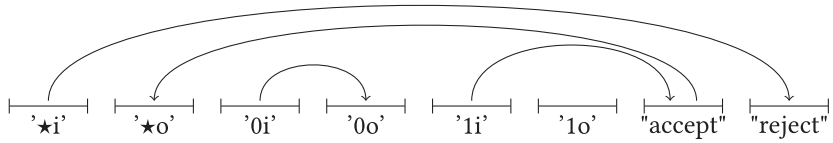
Fig. 2. Examples of words.

that all heads are repositioned to the left-hand endmarker at the end of the computation. We chose to allow for such an approximation, because the resulting machines are much simpler (one has to introduce additional states to perform the repositionning of heads) and therefore the examples easier to follow. The reader will convince herself that this does not change the fact that execution does represent the computation of the associated automata. The only difference can be seen in that accepting computation do not start and end at the same addresses (e.g., in the computation path shown on Figure 4(c)), which in turn makes them fail to be orthogonal to the simple test considered in this article.[7]

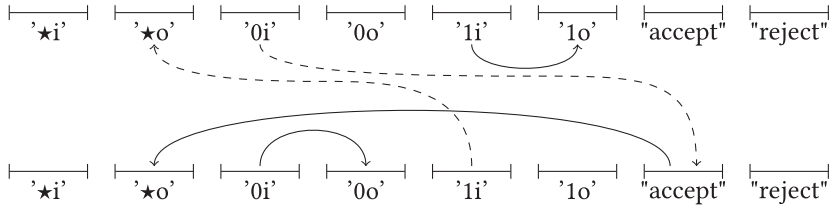
A First Machine. As a first example, let us represent an automata that accepts a word $\mathtt{w}$ if and only if $\mathtt{w}$ contains at least one symbol "1". This example was chosen because of the simplicity of the representation. Indeed, one can represent it by a graphing with a single state and that uses a single head (hence it is a graphing in the microcosm $\mathfrak{m}_1$). Its transition relation is then defined as follows: if the symbol read is a "1" then stop, if it is a "0", move the head to read the next symbol, if it is a "⋆" then reject. The representation as a graphing is shown in Figure 3(a). Notice that the symbol "⋆" plays both the role of left and right end-markers.

We then illustrate the computation of this graphing with the representations of the words ⋆0 (Figure 2(a)), ⋆11 (Figure 2(b)), and ⋆01 (Figure 2(c)) in Figure 4. The computation is illustrated by showing the two graphings (the machine and the integer), one on each side of the set of vertices. Notice that in those figures the graphing corresponding to the machine has been replaced by one of its *refinements* to reflect the splitting of the unit intervals induced by the representation of the words. The result of the computation is the set of alternating paths from the set of vertices $\{\mathsf{accept}, \mathsf{reject}\}$ to itself: in each case there is at most one, which is stressed by drawing the edges it is composed in boldface. The machine accepts the input if and only if there is a single path from

[7]Here, one could choose to consider a more complex notion of test that would allow for such behaviour; however, it is much easier to restrict the set of automata considered as it is clear that the restriction does not impact the set of languages recognised.



(a) The graphing representing the first machine.



(b) The graphing representing the second machine.

Fig. 3. Examples of machines: The plain lines are realised by simple translations, while the dashed lines are realised by a composition of a translation and the map s2.

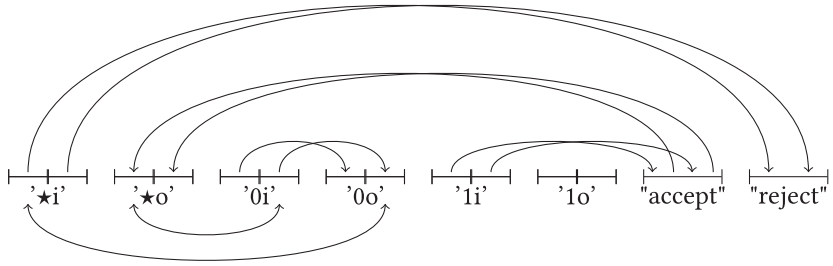
"accept" to "accept." One can see in the figure that this machine accepts the words ⋆11 and ⋆01 but not the word ⋆0.

Notice how the computation can be understood as a game between two players. We illustrate this on the computation on the word ⋆01. First the machine, through its edge from "accept" to "⋆o", asks the input "What's your first symbol?" The integer, through its edge from "⋆o" to "0i", answers "It's a '0'." Then, the machine asks "What's your next symbol?" (the edge from "0i" to "0o"), to which the integer replies "It's a '1'." (the edge from "0o" to "1i"). At this point, the machine accepts (the edge from "1i" to "accept").

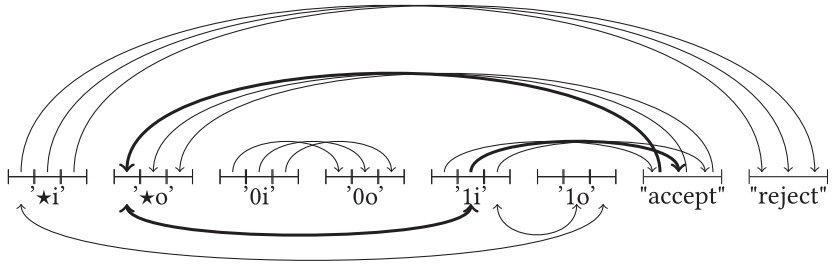
A Second Machine. We chose as a second example a more complex machine. The language it accepts is actually a regular language (the words that contain at least one symbol "1" and one symbol "0"), i.e., can be computed by a single-head automata. However, we chose to compute this language by first looking for a symbol "1" in the word and, if this part is successful, activate a second head looking for a symbol "0", i.e., we compute it with a graphing in the microcosm $\mathfrak{m}_2$ even if as a regular language it can be computed by graphings in $\mathfrak{m}_1$. This automata has two states corresponding to the two parts in the algorithmic procedure: a state "looking for a '1'" and a state "looking for a '0'"; we write these states "L1" and "L0", respectively. The graphing representing this automata is shown in Figure 3(b), where there are two sets of vertices one above the other: the row below shows the vertices in state L1 while the row on top shows the vertices in state L0. Notice the two dashed lines that are realised using a permutation and correspond to the change of principal head.[8]

We illustrate the computation of this machine with the same inputs ⋆0, ⋆11, and ⋆01 in Figure 6. Notice that to deal with states, the inputs are duplicated. Once again, acceptance corresponds with the existence of a path from the "accept" vertex to itself (without changing states). The first computation does not go far: the first head moves along the word and then stops as it did not encounter a symbol "1" (this path is shown by thick edges).

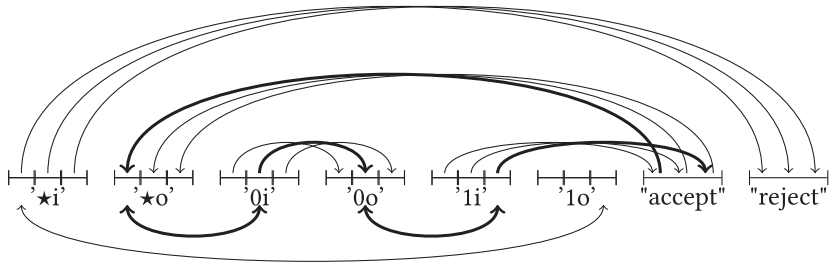
[8]The machine represented by graphings have a single active head at a given time. The permutations s_i ($i \leqslant 2$) swap the ith head with the first one, making the ith active. Reusing the permutation s_i then activates the former active head and "deactivates" the ith head.



(a) Computing with input ⋆0.



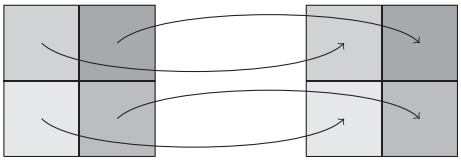
(b) Computing with input ⋆11.



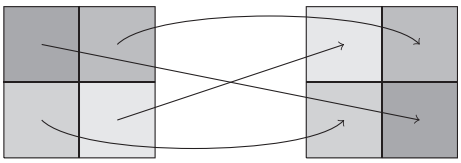
(c) Computing with input ⋆01.

Fig. 4. Example: Computing with the first machine.

The second computation is more interesting. First, it follows the computation of the previous machine on the same input: It asks for the first symbol, acknowledge that it is a "1", then activates the second head and changes state. At this point, the path continues as the second head moves along the input: this seems to contradict the picture, since our path seems to arrive on the middle splitting of the vertex "⋆o", which would not allow us to continue through the input edge whose source is the left-hand splitting of this same vertex. However, this is forgetting that we changed the active head. Indeed, what happens is that the permutation moves the splitting along a new direction: while the same transition realised as a simple translation would forbid us to continue the computation, realising it by the permutation s_2 allows the computation to continue. We illustrate in Figure 5 how the permutation actually interact with the splitting, in the case the interval is



(a) A simple translation



(b) A translation combined with a permutation

Fig. 5. How permutations interact with splittings.

split into two pieces. The representation of vertices as unit intervals is no longer sound as we are actually using the two first copies of $[0,1]$ in the Hilbert cube $[0,1]^{\mathbb{N}}$ to represent this computation, hence working with unit squares. In this case, however, the second head moves along the input without encountering a symbol "0" and then stops. This path is shown by thick edges in the figure.

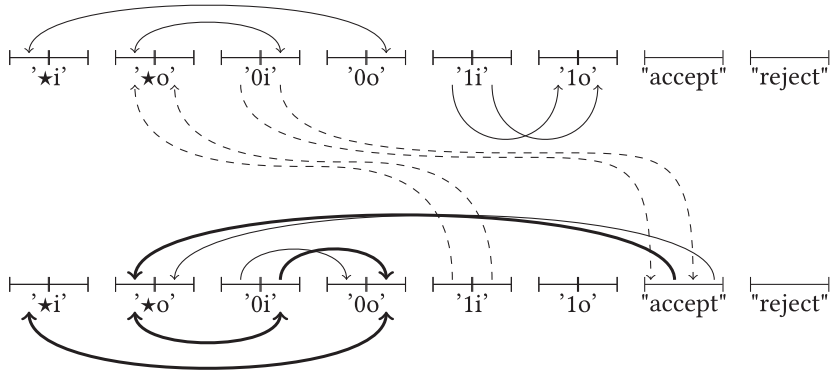
The last computation accepts the input. As in the previous case, the first head moves along the input, encounters a symbol "1", changes its state, and activates the second head. As in the previous case, the computation can continue at this point, and the second head encounters a "0" as the first symbol of the input. Then the machine accepts. This path is shown in the figure by thick edges.

4.3 Automata as Machines

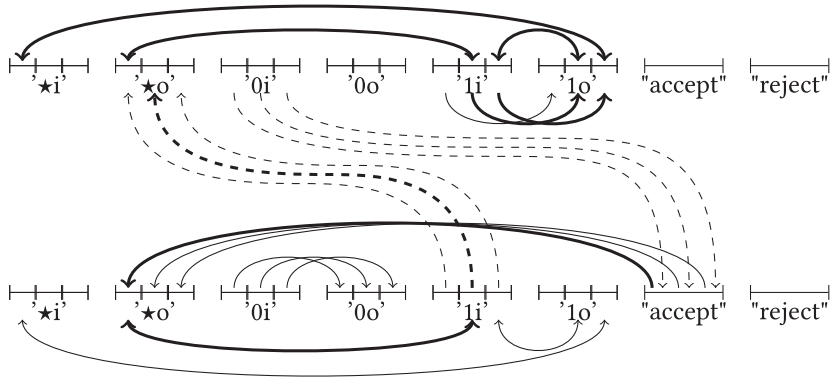
There are two main differences between the model of multihead automata with k heads and the notion of $\mathfrak{m}_k$-machines.

- The first difference is that when one "moves the ith head" of a $\mathfrak{m}_k$-machine, it induces a reindexing of the sets of heads. i.e., a $\mathfrak{m}_k$-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 $\bar{Q} = Q \times \mathfrak{G}_k$; the set of permutations $\mathfrak{G}_k$ being used to keep track of the heads' reindexings.
- The second difference comes from the fact that the computation of $\mathfrak{m}_k$-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*. That is, 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 $\mathtt{M}$ with a set of states Q will be realised as a $\mathfrak{m}_k$-machine with an extended set of states (encoded as the *dialect*) $\bar{Q} \times \{\star, 0, 1\}^k$.

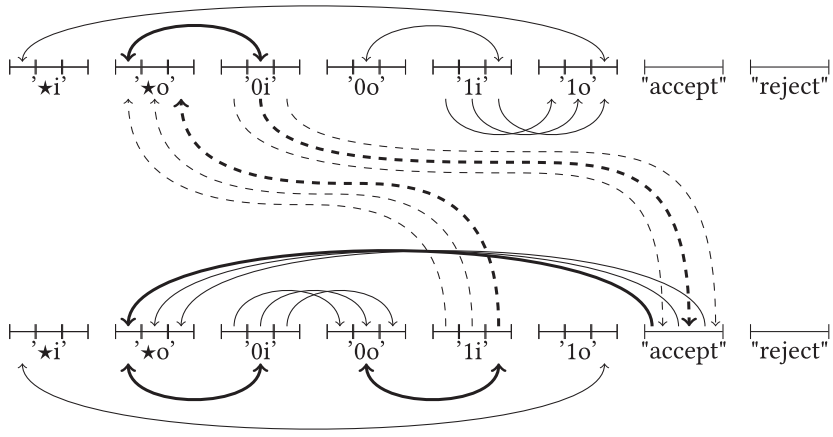
Definition 4.5. Let $\mathtt{M}$ be an automaton with k heads. We here write $\mathtt{M} = (\Sigma, Q, \to)$. We define $\{\mathtt{M}\}$ a graphing in $\mathfrak{m}_k$ with dialect – set of states – $Q \times \mathfrak{G}_k \times \{\star, 0, 1\}^k$ as follows.



(a) Computing with input ⋆0.



(b) Computing with input ⋆11.



(c) Computing with input ⋆01.

Fig. 6. Example: Computing with the second machine.

The set of edges of $\{\mathtt{M}\}$ is the set

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

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

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

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

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

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

Let us explain how this encoding simulates the automaton. We fix a word $\mathtt{w} = \star a_1 a_2 \dots a_n$ and a configuration $\mathtt{C}$ 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 $\mathtt{t} = (\vec{\mathrm{s}}, \mathrm{q}) \to (\mathrm{i}, \mathrm{d}', \mathrm{q}')$. There is a family of corresponding edges in the automaton, denoted by $(\{\mathtt{t}\}, \mathrm{a}, \mathrm{d}, \sigma)$. 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 (a, 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 $(\{\mathtt{t}\}, \mathrm{a}, \mathrm{d}, \sigma)$ maps the set $\langle (s,d) \rangle \times \{(q, \sigma, \vec{s})\}$ to $\langle (s_i, d') \rangle \times \{(q', \tau_{1,\sigma(i)} \circ \sigma, \vec{s}[s_{\sigma^{-1}(1)} := a])\}$ (supposing $q \neq \mathtt{init}$ and $q' \neq$ accept, 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 ith head adequately by reindexing it using the map $\mathsf{p}_{(1,\sigma(\mathrm{i}))}$ and waiting for the integer to provide its next value in direction d' by fixing the target subset $\langle (s_i, d') \rangle$ (s_i being the last value read by the ith head).

The following proposition is then proved by induction.

Proposition 4.6. *Let* $\mathtt{M}$ *be a* k*-head automaton. Alternating paths of odd length between* $\{\mathtt{M}\}$ *and* $!W_{\mathtt{w}}$ *of source* $\langle \mathsf{a} \rangle_{\mathrm{Y}}$ *(respectively, of source* $\langle \mathsf{r} \rangle_{\mathrm{Y}}$*) with*[9] $Y = [0, \frac{1}{\lg(\mathtt{w})}]^k \times [0,1]^{\mathbb{N}}$ *are in bijective correspondence with the non-empty computation traces of* $\mathtt{M}$ *given* $\mathtt{w}$ *as input.*

Corollary 4.7. *The automaton* $\mathtt{M}$ *accepts the word* $\mathtt{w}$ *if and only if there exists no alternating path between* $\{\mathtt{M}\}$ *and* $!W_{\mathtt{w}}$ *from* $\langle \mathsf{r} \rangle$ *to itself.*

Definition 4.8. We define the test $\mathcal{T}_{\neq}$ as the set consisting of the projects $\mathrm{t}^{\neq}_{\zeta} = (\zeta, \mathrm{Id}_{\langle \mathsf{r} \rangle})$, where $\zeta \neq 0$ and $\mathrm{Id}_{\langle \mathsf{r} \rangle}$ is the graphing with a single edge and trivial dialect $[0]$: $\{(\langle \mathsf{r} \rangle, 0, 0, \mathrm{x} \mapsto \mathrm{x}, 1 \cdot \mathbf{1})\}$.

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

[9] To understand where the subset Y comes from, we refer the reader to the proof of Lemma 4.14.

$\theta \circ \theta^{-1}$, which shows, by simplification, that there exists a corresponding path alternating between $!W$ and $\mathrm{Id}_{\langle \mathtt{r} \rangle}$.

Proposition 4.9. *The test $\mathcal{T}_{-}$ is uniform w.r.t. $\mathfrak{m}_\infty$-machines.*

Proposition 4.10. *Let $\mathtt{M}$ be a $\mathbf{2nfa}(k)$, $\mathtt{w}$ a word. Then $\mathtt{w} \in \mathcal{L}^{\mathcal{T}_{-}}(\{\mathtt{M}\})$ if and only if $\mathtt{M}$ accepts $\mathtt{w}$.*

Proof. From Proposition 4.6 and the constraint on automata that they should reinitialise their pointer to the left end-marker before accepting or rejecting, we know that $R = \{\mathtt{M}\} ::! W_{\mathtt{w}}$ 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 4.6 – as there are rejecting computation traces of $\mathtt{M}$ given $\mathtt{w}$ as input.

Moreover, $[\![\{\mathtt{M}\}, !W_{\mathtt{w}}]\!]_m$ 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 $\xi + [\![R, \{\mathrm{Id}_{\langle \mathtt{r} \rangle}\}]\!]_m \neq 0, \infty$ for all $\xi \neq 0$. Now, this is true if and only if $[\![R, \{\mathrm{Id}_{\langle \mathtt{r} \rangle}\}]\!]_m = 0$, 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 $\{\mathrm{Id}_{\langle \mathtt{r} \rangle}\}$ of weight $1 \cdot \mathbf{1}$. □

Theorem 4.11. *Any language computed by a k-head automaton is computed by a $\mathfrak{m}_k$-machine w.r.t. $\mathcal{T}_{-}$.*

4.4 Machines as Automata

We will now describe how one can define an 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 4.12. 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 4.13. *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 $\bar{M}$ such that, for all test $\mathcal{T}$, $\mathcal{L}^{\mathcal{T}}(M) = \mathcal{L}^{\mathcal{T}}(\bar{M})$.*

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, in our framework, to the so-called technical lemma, which was essential in previous work involving operator algebras [3, 4].

Lemma 4.14 (Technical Lemma). *Let M be a $\mathfrak{m}_\infty$-machine. The computation of M with a representation $!W$ of a word $\mathtt{w}$ is equivalent to the computation of a finite*[10] *Ω-weighted graph $\bar{M}$ and the graph representation $\bar{W}_{\mathtt{w}}$ of $\mathtt{w}$.*

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 $\mathtt{w} \in \Sigma^*$, write k the length of $\mathtt{w}$ and consider the project $(0, !W_{\mathtt{w}})$. Let us remark that all maps realising edges in M or in $!W_{\mathtt{w}}$ are of the form $\phi \times \mathrm{Id}_{\bigtimes_{i=N+1}^{\infty}[0,1]}$. We can therefore consider that the underlying space is $\mathbf{Z} \times [0,1]^N$ instead of $\mathbf{X}$ by just replacing realisers $\phi \times \mathrm{Id}_{\bigtimes_{i=N+1}^{\infty}[0,1]}$ by ϕ. Moreover, the maps ϕ here act either as permutations

[10] Whose size depends on both the length of $\mathtt{w}$ and the smallest k such that M is a $\mathfrak{m}_k$ machine.

over copies of $[0,1]$ (realisers of edges of M) or as permutations over a decomposition of $[0,1]$ into k intervals (realisers of $!W_{\mathsf{w}}$). Consequently, all realisers act as permutations over the set of N-cubes $\{\bigtimes_{i=1}^{N}[k_i/k,(k_i+1)/k] | 0 \leqslant k_i \leqslant k-1\}$, i.e., their restrictions to N-cubes are translations.

Consequently, one can build two (thick[11]) graphs $\bar{M}$ and $\bar{W}_{\mathsf{w}}$ over the set of vertices $\Sigma^{\Updownarrow} \times \{\bigtimes_{i=1}^{N}[k_i/k,(k_i+1)/k] | 0 \leqslant k_i \leqslant k-1\}$ 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 $\bigtimes_{i=1}^{N}[k_i/k,(k_i+1)/k]$ onto the N-cube $\bigtimes_{i=1}^{N}[k'_i/k,(k'_i+1)/k]$. There is an edge in $\bar{W}_{\mathsf{w}}$ 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 $i \geqslant 2$, and there is an edge in W_{w} of source $\langle s\rangle \times [k_1/k,(k_1+1)/k] \times [0,1]^{\mathbf{N}}$ and target $\langle s'\rangle \times [k'_1/k,(k'_1+1)/k] \times [0,1]^{\mathbf{N}}$.

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

This lemma will be useful because of the following proposition.

Proposition 4.15. *For any $\mathfrak{m}$-machine G and word representation $!W$, $G :: !W$ is orthogonal to $\mathcal{T}_{-}$ if and only if there are no cycles between G and $!W \otimes Id_{\langle \mathsf{r}\rangle}$ going through $\langle \mathsf{r}\rangle$.*

Proof. We use here the *trefoil property* for graphings [28], which in this case translates as $[\![(0,G)::(0,!W),(t,T)]\!]_m = [\![(0,G),(0,!W)::(t,T)]\!]_m$. 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 $(0,!W)\otimes(\zeta,\mathrm{Id}_{\langle \mathsf{r}\rangle})$ for all $\zeta \neq 0$. But $(0,!W)\otimes(\zeta,\mathrm{Id}_{\langle \mathsf{r}\rangle}) = (\zeta,!W\otimes \mathrm{Id}_{\langle \mathsf{r}\rangle})$. Thus $G::!W$ is orthogonal to $\mathcal{T}_{-}$ if and only if $\zeta + [\![G,!W\otimes \mathrm{Id}_{\langle \mathsf{r}\rangle}]\!]_m \neq 0,\infty$, i.e., if and only if there are no alternating cycles between G and $!W\otimes \mathrm{Id}_{\langle \mathsf{r}\rangle}$ of weight of the form $a\cdot\mathbf{1}$. Finally, since all weights in G and $!W$ are equal to 1, such cycles need to go through $\langle \mathsf{r}\rangle$. □

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 $\mathsf{p}_{\tau_{i,j}}$, where $\tau_{i,j}$ denotes the transposition exchanging i 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 $!W\otimes \mathrm{Id}_{\langle \mathsf{r}\rangle}$ starting in $\langle \mathsf{r}\rangle$, using the fact that this can be done by following the paths between finite graphs $\bar{M}$ and $\bar{W}_{\mathsf{w}}\otimes \mathrm{Id}$ using Lemma 4.14.

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

If $\Im$ 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 $\Im \neq \emptyset$. We will build an automaton $[G]$ whose set of states is equal to $Q\times \mathfrak{G}_N \times \Im \times \{\star,0,1\}^N$. 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 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 a transposition $\mathsf{p}_{\tau_{i,j}}$ 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.

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

We now define the transition relation of the automaton.

- Each edge e in M, of source $(\langle \mathsf{r}\rangle, \mathrm{q})$ and target $(\langle (s', d')\rangle, q')$ realised by $\tau_{1,j}$ is represented by the family of transitions $(\vec{a}, (q, \sigma, i, \vec{s})) \to (\sigma(j), d', (q', \tau_{1,j} \circ \sigma, i, \vec{s}))$ 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 $(\vec{a}, (q, \sigma, i, \vec{s}) \to (\sigma(j), d', (q', \tau_{1,j} \circ \sigma, i, \vec{s}))$ 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 $(\langle \mathsf{r}\rangle, \mathrm{q})$ with $q \in \Im$ realised by $\tau_{1,j}$ is represented by:
  —the family of transitions $(\vec{a}, (q, \sigma, i, \vec{s})) \to (\sigma(j), d', (q', \tau_{1,j} \circ \sigma, i, \vec{s}))$ for $\vec{a}$ such that $a_{\sigma(1)} = s$ and $a_{\sigma(j)} = s'$ and $q \neq i$;
  —the family of transitions $(\vec{a}, (q, \sigma, i, \vec{s})) \to \mathtt{reject}$ for $\vec{a} = \vec{s}$ and $i = q$;
- For each $i \in \Im$ and $\vec{s} \in \{\star, 0, 1\}^N$, there is a transition $(\vec{a}, \mathtt{init}) \to (\vec{a}, (\mathrm{i}, \mathrm{Id}, \mathrm{i}, \vec{a}))$.

Definition 4.16. For all integer N and Γ_N-essential $\mathfrak{m}_N$-machine G, we denote $[G]$ the N-head automaton described above.

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

Proposition 4.17. *Let G be a Γ_N-essential $\mathfrak{m}_N$-machine, $\mathtt{w}$ a word and $!W$ be a word representation of $\mathtt{w}$. There is an alternating cycle between $G :: !W$ and $Id_{\langle \mathsf{r}\rangle}$ going through $\langle \mathsf{r}\rangle$ if and only if the automaton $[G]$ rejects when given $\mathtt{w}$ as input.*

Theorem 4.18. *Any language computed by a $\mathfrak{m}_N$-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 $\mathfrak{m}_N$-machine. Then, there exists a Γ_N-essential $\mathfrak{m}_N$-machine H such that $\mathcal{L}^{\mathcal{T}_-}(G) = \mathcal{L}^{\mathcal{T}_-}(H)$. Now, we have defined the automaton $[H]$, which, by Proposition 4.17, rejects an input $\mathtt{w}$ if and only if there is an alternating path between H and $W_{\mathtt{w}} \otimes \mathrm{Id}_{\langle \mathsf{r}\rangle}$ going through $\langle \mathsf{r}\rangle$. But this is equivalent, by Proposition 4.15, to the fact that $H :: !W_{\mathtt{w}}$ is not orthogonal to $\mathcal{T}_-$. Summing up, we have shown that $\{\mathtt{M}\}$ rejects $\mathtt{w}$ if and only if $\mathtt{w} \notin \mathcal{L}^{\mathcal{T}_-}(G)$. □

5 CONCLUSION

Combining Theorems 4.11 and 4.18, we obtain the characterisation of the hierarchy of sublinear complexity classes announced in the introduction.

Theorem 5.1. *For all $i \in \mathbf{N}^* \cup \{\infty\}$, $\mathbf{Pred}(\mathfrak{m}_i) = \mathrm{co2Nfa(i)}$*

In particular, the microcosm $\mathfrak{m}_1$ characterises the class of regular languages, while the microcosm $\mathfrak{m}_\infty$ 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 [8]. 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 coNPtime. As explained in an

overview and perspective paper [24], the results will be adapted for deterministic and probabilistic classes.

Now, the result we obtained can be understood from two different perspectives, each one leading to specific directions for future work. First, the strong ties our construction has with linear logic and its constrained variants leads to the possibility of extracting from our characterisations a purely syntactic characterisation of logarithmic space computation. Second, the geometric nature of the characterisation opens the possibility of using tools and techniques from mathematics (and more precisely from the theory of ergodic theory) to be used in complexity theory.

5.1 The Logical View

As explained above, the set $\mathbf{Pred}(\mathfrak{m}_i)$ can be understood both as semantic restrictions over the set of computable predicates in the model of Elementary Linear Logic described by the microcosm $\mathfrak{p}$ (Theorem 3.5), and 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 does not allow us to view those as exponentiated objects themselves. Intuitively, the characterisation of coNLogspace obtained above should correspond 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. Future work include a study of such a logical system and whether it provides a constrained linear logic characterising the class Logspace.

5.2 The Geometric View

As explained in the Introduction and not developed yet, the results we obtain are of a geometrical nature. Indeed, the classes are here characterised by microcosms that are (in this case) actions of groups on a measured space by measure-preserving maps. Indeed, the microcosm $\mathfrak{m}_i$ is obtained from the set of translations on $\mathbf{Z}$ together with the set of maps induced by the action of the set $\mathfrak{G}_i$ 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 article: this is because they are need to interact with the input. Therefore, only the action of the group $\mathfrak{G}_i$ 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 article can be shown to be non-homotopic by using mathematical invariants [10]. Together with the separation result (Theorem 4.3), this leads the author to the conjecture that the converse holds [23, 24], i.e., that non-equivalent group actions yield distinct complexity classes.

REFERENCES

[1] Clément Aubert, Marc Bagnol, Paolo Pistone, and Thomas Seiller. 2014. Logic programming and logarithmic space. In *Proceedings of the 12th Asian Symposium on Programming Languages and Systems (APLAS'14) (Lecture Notes in Computer Science)*, Jacques Garrigue (Ed.), Vol. 8858. Springer, 39–57. DOI: http://dx.doi.org/10.1007/978-3-319-12736-1_3

[2] Clément Aubert, Marc Bagnol, and Thomas Seiller. 2016. Unary resolution: Characterizing Ptime. In *Proceedings of the International Conference on Foundations of Software Science and Computation Structures (FOSSACS'16)*.

[3] Clément Aubert and Thomas Seiller. 2016a. Characterizing co-NL by a group action. *Math. Struct. Comput. Sci.* 26, 4 (2016), 606–638. DOI: http://dx.doi.org/10.1017/S0960129514000267

[4] Clément Aubert and Thomas Seiller. 2016b. Logarithmic space and permutations. *Info. Comput.* 248 (2016), 2–21. DOI: http://dx.doi.org/10.1016/j.ic.2014.01.018

[5] Patrick Baillot. 2011. Elementary linear logic revisited for polynomial time and an exponential time hierarchy. In *Proceedings of the Asian Symposium on Programming Languages and Systems (APLAS'11) (Lecture Notes in Computer Science)*, Hongseok Yang (Ed.), Vol. 7078. Springer, 337–352. DOI: http://dx.doi.org/10.1007/978-3-642-25318-8_25

[6] Theodore Baker, John Gill, and Robert Solovay. 1975. Relativizations of the P $\stackrel{?}{=}$ NP question. *SIAM J. Comput.* 4, 4 (1975), 431–442. DOI: http://dx.doi.org/10.1137/0204037

[7] Stephen Bellantoni and Stephen Cook. 1992. A new recursion-theoretic characterization of the polytime functions. *Comput. Complex.* 2 (1992). Retrieved from https://doi.org/10.1007/BF01201998.

[8] Stephen A. Cook. 1971. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM* 18, 1 (Jan. 1971), 4–18. DOI: http://dx.doi.org/10.1145/321623.321625

[9] Vincent Danos and Jean-Baptiste Joinet. 2003. Linear logic & elementary time. *Info. Comput.* 183, 1 (2003), 123–137. DOI: http://dx.doi.org/10.1016/S0890-5401(03)00010-5

[10] Damien Gaboriau. 2000. Coût des relations d'équivalence et des groupes. *Inventiones Mathematicae* 139 (2000), 41–98. DOI: http://dx.doi.org/10.1007/s002229900019

[11] Jean-Yves Girard. 1987. Linear logic. *Theor. Comput. Sci.* 50, 1 (1987), 1–101. DOI: http://dx.doi.org/10.1016/0304-3975(87)90045-4

[12] Jean-Yves Girard. 2011. Geometry of interaction V: Logic in the hyperfinite factor.*Theor. Comput. Sci.* 412 (2011), 1860–1883. DOI: http://dx.doi.org/10.1016/j.tcs.2010.12.016

[13] Jean-Yves Girard. 2012. Normativity in logic. In *Epistemology versus Ontology*, Peter Dybjer, Sten Lindström, Erik Palmgren, and Göran Sundholm (Eds.). Logic, Epistemology, and the Unity of Science, Vol. 27. Springer, 243–263. DOI: http://dx.doi.org/10.1007/978-94-007-4435-6_12

[14] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. 1992. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.* 97, 1 (April 1992), 1–66. DOI: http://dx.doi.org/10.1016/0304-3975(92)90386-T

[15] Martin Hyland and Andrea Schalk. 2003. Gluing and orthogonality for models of linear logic. *Theor. Comput. Sci.* 294 (2003). DOI: http://dx.doi.org/10.1016/S0304-3975(01)00241-9

[16] D. Leivant and J.-Y. Marion. 1993. Lambda calculus characterizations of poly-time. *Fundam. Info.* 19 (1993).

[17] Daniel Leivant and Jean-Yves Marion. 1994. Ramified recurrence and computational complexity II: Substitution and poly-space. *Lecture Notes Comput. Sci.* 933 (1994). DOI: http://dx.doi.org/10.1007/BFb0022277

[18] Buckhard Monien. 1976. Transformational methods and their application to complexity problems. *Acta Informatica* 6 (1976), 95–108.

[19] Alberto Naibo, Mattia Petrolo, and Thomas Seiller. 2016. On the computational meaning of axioms. In *Epistemology, Knowledge and the Impact of Interaction*. Springer, 141–184. DOI: http://dx.doi.org/10.1007/978-3-319-26506-3_5

[20] A. A. Razborov and S. Rudich. 1997. Natural proofs. *J. Comput. Syst. Sci.* 55 (1997). DOI: http://dx.doi.org/10.1006/jcss.1997.1494

[21] Thomas Seiller. 2012a. Interaction graphs: Multiplicatives. *Ann. Pure Appl. Logic* 163 (Dec. 2012), 1808–1837. DOI: http://dx.doi.org/10.1016/j.apal.2012.04.005

[22] Thomas Seiller. 2012b. *Logique Dans Le Facteur Hyperfini : Géometrie De L'interaction Et Complexité*. Ph.D. Dissertation. Université Aix-Marseille. Retrieved from http://tel.archives-ouvertes.fr/tel-00768403/.

[23] Thomas Seiller. 2015a. Measurable preorders and complexity. In *Proceedings of the Topology, Algebra and Categories in Logic Conference*.

[24] Thomas Seiller. 2015b. Towards a *Complexity-through-Realizability* Theory. Retrieved from http://arxiv.org/pdf/1502.01257.

[25] Thomas Seiller. 2016a. Interaction graphs: Additives. *Ann. Pure Appl. Logic* 167 (2016), 95–154. DOI: http://dx.doi.org/10.1016/j.apal.2015.10.001

[26] Thomas Seiller. 2016b. Interaction graphs: Full linear logic. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16)*. ACM, New York, NY, USA, 427–436. DOI: https://doi.org/10.1145/2933575.2934568

[27] Thomas Seiller. 2017a. Interaction graphs: Exponentials. Submitted. Retrieved from http://arxiv.org/pdf/1312.1094.
[28] Thomas Seiller. 2017b. Interaction graphs: Graphings. *Ann. Pure Appl. Logic* 168, 2 (2017), 278–320. DOI: http://dx.doi.org/10.1016/j.apal.2016.10.007
[29] Thomas Seiller. 2018. A correspondence between maximal Abelian sub-algebras and linear logic fragments. *Math. Struct. Comput. Sci.* 28, 1 (2018), 77–139. DOI: http://dx.doi.org/10.1017/S0960129516000062

Received September 2016; revised October 2017; accepted May 2018