Implicit complexity through linear realisability: polynomial time and probabilistic classes

THOMAS SEILLER*, CNRS, France

Based on work on realisability models for linear logic, the author recently proposed a new approach of implicit computational complexity. He showed how to characterise in this way a hierarchy of sub-linear space non-deterministic complexity classes by means of group actions. These classes, defined by means of two-way non-deterministic automata, range from regular languages to NLogspace (non-deterministic logarithmic space).

In the present paper, we extend those results in two directions. First, we show how the same techniques can be used to characterise the sub-linear space complexity classes defined by both deterministic and probabilistic two-way multi-head automata. We thus obtain characterisations of deterministic complexity classes between regular languages and Logspace, as well as a hierarchy of probabilistic classes between stochastic languages and PLogspace (unbounded error probabilistic logarithmic space). Second, we exhibit a monoid action capturing polynomial time computation based on pushdown machines, characterising both Ptime and PPtime (unbounded error probabilistic polynomial time).

1 INTRODUCTION

Complexity theory finds its root in three different papers that, in the span of a single year, tackled the difficult question of defining a notion of *feasible* computation [8, 12, 18]. The field of complexity theory then quickly developed, aiming at the definition and classification of functions based on how much resources (e.g. time, space) are needed to compute them. While progress on the classification aspects was quick in the early days, new results started to become scarcer and scarcer. The difficulty of the classification problem can be explained in several ways. First, from a logical point of view, the question of showing whether a complexity class cannot contain a given function corresponds to showing the negation of an existential statement. But the severe difficulty of this problem can be understood through negative results known as *barriers* [1, 6, 26], i.e. results stating that currently known methods cannot solve current open problems.

Implicit Computational Complexity (icc) aims at studying computational complexity only in terms of restrictions of languages and computational principles, for instance considering restrictions on recursion schemes, and was established by Bellantoni and Cook' landmark paper [7], and following work by Leivant and Marion [23, 24]. Amongst the different approaches to icc, several results use Girard's *linear logic* [15], a refinement of intuitionnistic logic which accounts for the notion of resource. 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 capturing complexity classes, e.g. the class Ptime is characterised by bll [17], sll [22] and dlal [2, 5], while Kalmar's elementary functions [21] are captured by ell [11].

The current work is part of a large programme aiming at developing a new approach to implicit computation complexity [36, 37, 39], based on linear realisability models – realisability models of linear logic. While standard icc approaches cannot lead to new separation results, this alternative

*T. Seiller was partially supported by the European Commission Marie Skłodowska-Curie Individual Fellowship (H2020-MSCA-IF-2014) project 659920 - ReACT, the CNRS grants BiGRE and LoBE, the DIM RFSI Exploratory project CoHOp, and the ANR ANR-22-CE48-0003-01 project DySCo.

Author's address: Thomas Seiller, LIPN – UMR 7030 CNRS & University Sorbonne Paris North, CNRS, 99, avenue Jean-Baptiste Clément, Villetaneuse, 93430, France, thomas.seiller@cnrs.fr.

semantic variant of ICC opens the way to use invariants from dynamical systems to prove complexity classes are not equal.

The difference with previous approaches lies in the use of types. ICC traditionally use typing systems as constraints excluding some programs: any behaviour that could lead to, say, superpolynomial execution time, will be forbidden by the type system. As a consequence, no program with a super-polynomial running time will be typable. Moreover, if the type system is well-designed it remains *complete*: any polynomial time computable function can be computed by a program following the typing discipline. But this result is *extensional*, and the majority of polynomial time algorithms are in fact not typable in the considered systems, making the approach unusable to tackle open separation problems. In realisability models, one starts from a model of computation, and no further constraints are imposed. Types are understood as *descriptors* rather than constraints: a program will be of type $A \to B$ as long as it produces an element of type B when given an input of type A. The constraints will therefore come from the model of computation considered, not the types.

Recently, the author proposed to exploit realisability models to provide new proof methods for separation [30, 31, 37]. In essence, the guiding intuition is that a computation should be mathematically modelled as a dynamical system, in the same way physical phenomena are. Obviously, while a computation (i.e. a run of a program) is deterministic and can be represented as such, a *program* is not in general: it might be e.g. probabilistic, deterministic, and may represent in itself several possible runs on a given input. His proposal is therefore to work with generalisations of dynamical systems introduced under the name of *graphings*, which is used to abstract the notion of transition function. Graphings are furthermore induced by a monoid action $\alpha : M \curvearrowright X$, a monoid homomorphism from M to endomorphisms of X, representing a model of computation: X represents the space of *configurations* of a machine (e.g. for Turing machines, the contents of the tape and position of the head), and M is generated by a set of basic *instructions* (e.g. such as "move the head to the right"). The overall method then relies on a theorem [35] stating that the collection of graphings induced by a monoid action onto a space $\alpha : M \curvearrowright X$ gives rise to a model of (fragments of) linear logic.

More specifically, the characterisations are obtained as follows [37]. Consider a monoid action α that will be used to characterise a complexity class. One picks a monoid action β extending α (i.e. β is defined from α by adding endormorphisms) such that the logic induced by β is at least Elementary Linear Logic (ELL). This allows for the definition, in the induced model, of the type $!\mathbf{Nat}_2 \multimap \mathbf{Bool}$[1] of β-graphings (programs in the model of computation described by β) that compute predicates over binary strings. As β extends α, one can further consider the set of α-graphings in this type $!\mathbf{Nat}_2 \multimap \mathbf{Bool}$. These are the programs computing predicates over binary strings and typable in the logic induced by α (which is less expressive than ELL): the set of those predicates is the complexity class characterised by α.

The first formal result obtained in this way [37] provided a correspondence between a hierarchy of group actions $\mathfrak{m}_1, \mathfrak{m}_2, \ldots, \mathfrak{m}_k, \ldots, \mathfrak{m}_\infty$ and a hierarchy of non-deterministic complexity classes between (and including) REGULAR– the class of regular languages – and coNLOGSPACE. We insist on the fact that this result characterises coNLOGSPACE– even though it is know to be equal to NLOGSPACE– because the technique developed captures the notion of acceptance of the former. Moreover, although both regular languages and NLOGSPACE are closed under complementation, this is not known for the intermediate classes characterised. Choosing different monoid actions lead to models in which the represented programs are of limited complexity, formalising an intuition that already appeared in the more involved context of operator algebras [36].

[1] In practice, the type considered is rather $!\mathbf{Nat}_2 \multimap \mathbf{NBool}$ where **NBool** is a non-deterministic version of **Bool**.

Monoid Action	deterministic model	non-deterministic model		probabilistic model
$\mathfrak{m}_1$	Regular	Regular	Regular	Stochastic
⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_k$	D_k	N_k	co-N_k	P_k
⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_\infty$	Logspace	NLogspace	coNLogspace	PLogspace
$\mathfrak{n}_\infty$	Ptime	Ptime	Ptime	PPtime

Fig. 1. Known characterisations. Contributions of the current paper are shown in blue cells.
Here, D_k (resp. N_k, P_k) is the class of languages decided by two-way k-heads (resp. nondeterministic, probabilistic) automata. Moreover, we distinguish two columns in the non-deterministic case as different tests lead to different characterisations. Lastly, we distinguish NLogspace and coNLogspace despite the Immerman–Szelepcsényi theorem.

One hope for complexity theory is that the equivalence between monoid actions defined as inducing the same complexity class could be formally related to standard equivalences of monoid/group actions, such as orbit equivalence. More precisely, one can ask the following question: is it true that if two actions α and β are *not* orbit equivalent, then they characterise different complexity classes? This would enable the use of invariants for orbit equivalence – such as ℓ^2-Betti numbers [14] or *cost* [13] – to obtain separation results (i.e. showing that two complexity classes are different). This approach to separation is further strengthened by two recent results. The first [38] relates the notion of *orthogonality* – which is used to define types – with zeta functions of dynamical systems. Defined in terms of (finite) orbits, the zeta function of a dynamical system is an invariant for orbit equivalence. The second result [39] reformulates and strengthens lower bounds results for algebraic models of computation by using graphings to exploit topological entropy – an invariant for conjugacy: an equivalence of actions finer than orbit equivalence.

Contributions. The current paper extends the previous characterisations to the corresponding deterministic, non-deterministic (with the notion of acceptance of NLogspace), and probabilistic hierarchies, at the same time capturing the polynomial time constraint. This is an important step in the overall program, as it shows the techniques apply to several computational paradigms, and extend to superlinear space complexity classes. As such, it puts current open problems in complexity, such as Logspace $\stackrel{?}{=}$ Ptime, within reach of the potential separation techniques mentioned above. From a more general point of view, the techniques provides the first Curry-Howard implicit characterisations[2] of probabilistic complexity classes, such as PLogspace (resp. PPtime) of problems decidable (with unbounded error) by a probabilistic machine using logarithmic space (resp. polynomial time) in the input. Figure 1 recapitulates[3] the known characterisations using the proposed approach, showing the results of the current paper in blue cells.

2 INTERACTION GRAPHS MODELS

Interaction Graphs (ig) models of linear logic were developed in order to generalise Girard's geometry of interaction (goi) constructions to account for *quantitative* aspects, in particular adapting to non-deterministic and probabilistic settings. The aim of the goi program approach is to obtain a *dynamic* model of proofs and their cut-elimination procedure, i.e. a semantics in which

[2]Some implicit characterisation of a few probabilistic classes already exist [10, 20], but they are not based on the Curry-Howard correspondence and do not apply to such a large extent of classes.

[3]This table is not exhaustive, but shows the most common classes. In particular, this paper also characterises numerous classes not shown here, notably the classes of languages recognized by k-head two-way atutomata with a pushdown stack where k is a fixed integer.

a program P applied to an argument *a do not* possess the same interpretation as the result of the computation (as opposed to denotational semantics). As a consequence, GOI (and hence IG) models comprise a mathematical counterpart of cut-elimination (or equivalently, program execution). One key insight of the author's work on IG models is that this mathematical counterpart, called the *execution formula* by Girard, can be understood as computing finite orbits in a dynamical system. We provide here a quick overview of the major concepts used in the paper but refer to previous work for full details [27, 35] and illustrations [31].

2.1 Abstract models of computation

In the general setting, graphings can be defined in many different flavours: discrete, topological, measurable, etc. In this paper, we will be working with measurable graphings, and will refer to them simply as *graphings*. Graphings act on a chosen space (hence here, on a measured space); the definition of graphings makes sense for any measured space $\mathbf{X}$, and under some mild assumptions on $\mathbf{X}$ it provides a model of (at least) Multiplicative-Additive Linear Logic (MALL) [33–35].

In order to define an Interaction graph model, two elements are needed. The first is a space $\mathbf{X}$. In full generality, this space could be discrete, a topological or measured space, a (topological) vector space, etc. In this paper, we will only consider measured spaces. This space intuitively corresponds to a space of *configurations* of the machines in the model of computation considered. The second element is a monoid action $\alpha : M \curvearrowright \mathbf{X}$. In practice, this monoid action can be described through the action of a set of generators I. I.e. each generator defines an endomorphism of $\mathbf{X}$ and the collection of endomorphisms $\{\alpha(m) \mid m \in I\}$ is enough to recover the full action α.

Example 1. As an example, let us consider Turing machines with a single tape. A natural representation of the model of computation would be an action of the basic instructions of a Turing machine (writing a symbol and moving the head on the tape) on a space $\mathbf{X}$ representing the possible configurations of a Turing machine. Once could for instance choose $\mathbf{X}$ as the set $\{0, 1, \star\}^{\mathbf{Z}}_{\star}$ of $\mathbf{Z}$-indexed sequences in $\{0, 1, \star\}$ that are almost-always equal to $\star$ (which is understood as representing blank cells on the tape). With the convention that the head of the Turing machine is located above the 0-th indexed element of the sequence, the basic instructions of a Turing machines give rise to the following maps:

- $\texttt{right} : (s_i) \mapsto (t_i)$ with $t_i = s_{i+1}$ for all i;
- $\texttt{left} : (s_i) \mapsto (t_i)$ with $t_i = s_{i-1}$ for all i;
- $\texttt{write}_* : (s_i) \mapsto (t_i)$ $(* \in \{0, 1, \star\})$ with $t_0 = *$ and $t_i = s_i$ for $i \neq 0$;

We now fix the measure space of interest in this paper.

Definition 2 (The Space)**.** We define the measure space $\mathbf{X} = \mathbf{R} \times [0, 1]^{\mathbf{N}} \times \{\star, 0, 1\}^{\mathbf{N}}$ where $\mathbf{R} \times [0, 1]^{\mathbf{N}}$ is considered with its usual Borel σ-algebra and Lebesgue measure. The space $\{\star, 0, 1\}^{\mathbf{N}}$ is endowed with the natural topology[4], the corresponding Borel σ-algebra and the natural measure given by $\mu(V(w)) = 3^{-\lg(w)}$.

Borrowing the notation introduced in earlier papers [33, 37], we denote by $(x, \mathbf{s}, \pi)$ the points in $\mathbf{X}$, where $\mathbf{s}$ and π are sequences for which we allow a concatenation-based notation, e.g. we write $a \cdot \mathbf{s}$ for the sequences whose first element is a. 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 $\mathbf{s}$.

Graphings are then defined as objects acting on the measured space $\mathbf{X}$. A parameter in the construction allows one to consider subsets of graphings based on *how* they act on the space. To do

[4] I.e. the topology induced by basic *cylindrical* open sets $V(w) = \{f : \mathbf{N} \to \{0, 1\} \mid \forall i \in [n], f(i) = w_i\}$ where w is a finite word of length n on the alphabet $\{0, 1\}$.

this, we fix a monoid action by measurable maps that abstractly describes a model of computation. Again, while graphings can be defined in full generality, some conditions on the chosen actions are needed to construct models of MALL [35]. The following monoid actions, which are of of interest in this paper, do satisfy these additional requirements.

Definition 3 (Monoid actions)**.** For all integer $i \geqslant 1$, we define the translations

$$\mathsf{t}_{\mathrm{z}} : (\mathrm{x}, \mathbf{s}, \pi) \mapsto (\mathrm{x} + \mathrm{z}, \mathbf{s}, \pi).$$

For all integer $i \geqslant 1$, and all bijection $\sigma : \mathbf{N} \to \mathbf{N}$ such that $\sigma(k) = k$ for all $k > i$, we define the maps

$$\mathsf{p}_{\sigma} : (\mathrm{x}, \mathbf{s}, \pi) \mapsto (\mathrm{x}, \sigma(\mathbf{s}), \pi).$$

We denote by $\mathfrak{m}_i$ the monoid generated by those maps, and by $\mathfrak{m}_\infty$ the union $\cup_{i>1}\mathfrak{m}_i$.

We also consider the following maps:

$$\begin{aligned}
\mathsf{pop} &: (x, \mathbf{s}, c \cdot \pi) \mapsto (x, \mathbf{s}, \pi),\\
\mathsf{push}_0 &: (x, \mathbf{s}, \pi) \mapsto (x, \mathbf{s}, 0 \cdot \pi),\\
\mathsf{push}_1 &: (x, \mathbf{s}, \pi) \mapsto (x, \mathbf{s}, 1 \cdot \pi),\\
\mathsf{push}_\star &: (x, \mathbf{s}, \pi) \mapsto (x, \mathbf{s}, \star \cdot \pi).
\end{aligned}$$

We denote by $\mathfrak{n}_i$ the monoid generated the monoid action $\mathfrak{m}_i$ extended by those, and by $\mathfrak{n}_\infty$ the union $\cup_{i>1}\mathfrak{n}_i$.

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

2.2 Programs as Graphings

We are now able to define graphings. Suppose given a monoid action $\alpha : M \curvearrowright \mathbf{X}$. Graphings are formally defined as quotients of graph-like objects called *graphing representatives*. Graphing representatives are families of weighted *edges*, similarly to a graph. Intuitively, a weighted edge is a triple (S, m, ω) where S is a subspace of $\mathbf{X}$, m is an element of the monoid M, and ω is a weight. Formally, the definition is more involved since one need to introduce a notion of *control states*: a graphing representative therefore comes with an additional *stateset* S, and edges have a fourth component $q \to q'$ with $q, q' \in S$. While the formal definition is involved, one can intuitively think of a graphing representative as a transition graph.

Example 4. In the example of Turing machines above, we define the following subspaces:

$$\mathbf{X}_a = \{(s_i) \in \mathbf{X} \mid s_0 = a\} \ (a \in \{0, 1, \star\}).$$

Note that a configuration belongs to $\mathbf{X}_i$ if and only if the head currently reads the symbol i. The following graphing (with stateset $\{\mathrm{even}, \mathrm{odd}, \mathrm{accept}, \mathrm{reject}\}$) represents a Turing machine that accepts inputs with an even number of 0 before the first $\star$ symbol:

$$\begin{aligned}
&\{(\mathbf{X}_0, \mathtt{right}, 1\mathrm{even} \to \mathrm{odd}), && (\mathbf{X}_0, \mathtt{right}, 1, \mathrm{odd} \to \mathrm{even}),\\
&(\mathbf{X}_1, \mathtt{right}, 1\mathrm{even} \to \mathrm{even}), && (\mathbf{X}_1, \mathtt{right}, 1\mathrm{odd} \to \mathrm{odd}),\\
&(\mathbf{X}_\star, \mathtt{right}, 1\mathrm{even} \to \mathrm{accept}), && (\mathbf{X}_\star, \mathtt{right}, 1\mathrm{odd} \to \mathrm{reject})\}
\end{aligned}$$

As in previouss work [37], we fix the monoid of weights of graphings Ω to be equal to $[0, 1]\times\{0, 1\}$ with usual multiplication on the unit interval and the product on $\{0, 1\}$.

Notation 5. 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 $m(x, y) = xy$ (used in Theorem 11).

We are now ready to give the formal definition of graphing representatives.

Definition 6 (Graphing representative)**.** Let $\alpha : M \curvearrowright \mathbf{X}$ be a monoid action, V^G a measurable subset of $\mathbf{X}$, and S^G a finite set. A (Ω-weighted) *α-graphing representative* G of *support* V^G and *stateset* D^G is a family:

$$\{(S_e^G, m_e^G, \omega_e^G, \mathtt{i}_\mathrm{e}^\mathrm{G} \to \mathtt{o}_\mathrm{e}^\mathrm{G}) \mid \mathrm{e} \in \mathrm{E}^\mathrm{G}\},$$

where:

- S_e^G is a measurable subspace[5] of $V^G \times D^G$;
- m_e^G is an element of M such that $\phi_e^G(S_e^G) \subseteq V^G$;
- $\omega_e^G \in \Omega$ is a *weight*;
- and $\mathtt{i}_\mathrm{e}^\mathrm{G}, \mathtt{o}_\mathrm{e}^\mathrm{G}$ are elements of D^G.

We will refer to elements of E^G as *edges*. For any edge $e \in E^G$ the set $S_e^G \times \{\mathtt{i}_\mathrm{e}^\mathrm{G}\}$ is called the *source* of e, and the set $T_e^G \times \{\mathtt{o}_\mathrm{e}^\mathrm{G}\}$ where $T_e^G = \phi_e^G(S_e^G)$ is called the *target* of e.

The notion of graphing representatives captures both an action on a space and a specific representation of it. As a simple example, suppose given two subspace S and S' such that $S \cap S'$ is negligible. Then the graphing representatives $H = \{(S \cup S', m, \omega, \mathtt{i} \to \mathtt{o})\}$ and $H' = \{(S, m, \omega, \mathtt{i} \to \mathtt{o}), (\mathrm{S}', \mathrm{m}, \omega, \mathtt{i} \to \mathtt{o})\}$ somehow represent the same action on X in two different ways. The notion of graphing is obtained by dissociating the action from the specific representation.

In earlier work [35], the equivalence relation used in the quotient was defined through a notion of *refinement*. To ease the presentation, we here provide a more direct reformulation.

Definition 7 (Graphing)**.** We define on graphing representatives the equivalence defined from:

$$\{(S \cup S', m, \omega, \mathtt{i} \to \mathtt{o})\} \text{ and } S \cap S' =_{a.e.} \emptyset \\ \sim \{(S, m, \omega, \mathtt{i} \to \mathtt{o}), (\mathrm{S}', \mathrm{m}, \omega, \mathtt{i} \to \mathtt{o})\},$$

by contextual closure – i.e. if $H \sim H'$, then $G \cup H \sim G \cup H'$.

A *graphing* is an equivalence class of graphing representatives w.r.t. this equivalence.

Since all operations considered on graphings were shown to be compatible with this quotienting [35], i.e. well defined on the equivalence classes, we will in the following make no distinction between a graphing – as an equivalence class – and a graphing representative belonging to this equivalence class.

Remark 8. As stated in a recent paper [38], the set of *deterministic* (resp. *probabilistic*) α-graphings is in one-to-one correspondance with partial dynamical systems (resp. discrete-image sub-probability kernels) $f : X \rightharpoonup X$ whose graph $\{(x, f(x)) \mid x \in \mathrm{dom}(f)\}$ is included in the pre-order $\mathcal{P}(\alpha) = \{(x, y) \mid \exists m \in M, \alpha(m)(x) = y\}$, which generalises the notion of Borel equivalence relation [36].

2.3 Linear Realisability models

The author showed that, under mild hypotheses on the monoid action that are satisfied in the examples considered here, the set of α-graphings assemble into realisability models for (fragments of) linear logic. This construction is based on two notions: *execution*, an operation which represents cut-elimination (or equivalently, the execution of programs), and *orthogonality*, a binary relation accounting for linear negation.

Execution is defined in terms of alternating paths. Given two graphings F, G, one defines their execution as the graphing of maximal alternating paths between them. In terms of the corresponding

[5] 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.

dynamical systems f, g – when those graphings are deterministic –, this defines the set of maximal orbits in the composition $g \circ f$. The following definition, though involved, describes a graphing representative of this set in full generality.

Notation 9. We denote $\mathrm{AltPath}(F, G)$ the set of alternating path between F and G, i.e. the set of paths $\pi = e_1 e_2 \dots e_n$ such that for all $i = 1, \dots, n-1$, $e_i \in F$ iff $e_{i+1} \in G$. Given a path $\pi = e_1 e_2 \dots e_n$, we define $\phi_\pi = \phi_{e_1} \circ \phi_{e_2} \circ \dots \circ \phi_{e_n}$, its weight $\omega_\pi = \prod_{i=1}^{n} \omega_{e_i}$, and its *domain* S_π – the maximal subspace on which ϕ_π is well-defined.

Definition 10 (Execution)**.** Let F and G be graphings of respective supports $V^F = V \uplus C$ and $V^G = C \uplus W$ with $V \cap W$ of null measure, and take representatives such that the source and target of all edges are either entirely included in C or do not intersect C. Their *execution* $F :: G$ is the graphing of support $V \uplus W$ and stateset $D^F \times D^G$ defined as:

$$\begin{aligned} F :: G \quad = \quad & \big\{ \big(S_\pi, \phi_\pi, \omega_\pi, (\mathtt{i}_{\mathrm{e}_1}, \mathtt{i}_{\mathrm{e}_2}) \to (\mathtt{o}_{\mathrm{e}_{n-1}}, \mathtt{o}_{\mathrm{e}_n})\big) \\ & \qquad \mid \pi = e_1, e_2, \dots, e_n \in \mathrm{AltPath}(F, G) \big\} . \end{aligned}$$

We now recall the notion of measurement. When restricted to the monoid actions considered in this paper, the expression of the measurement can be simplified. We therefore only give here this simpler expression and point the curious reader to earlier work for the general case [35].

Definition 11. The measurement between two graphings is defined as

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

where $\rho_{\phi_\pi(x)} = \inf\{n \in \mathbf{N} \mid \phi_\pi^n(x) = x\}$ (here $\inf \emptyset = \infty$), $\mathrm{Cycles}(F, G)$ is the set of alternating cycles[6] between F and G, and the support $\mathrm{supp}(\pi)$ of π is the set of points x belonging to a finite orbit [35, Definition 41].

The measurement is used to define linear negation. But first, let us recall the notion of *project* which is the semantic equivalent of *proofs*, and uses formal sums [32].

Definition 12. 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.

We can then define an *orthogonality relation* on the set of projects. Orthogonality captures the notion of linear negation and somehow translates the correctness criterion for proof nets. Its definition is based on the measurement defined above, extended to formal weighted sums of graphings by "linearity" [32, 35].

Definition 13. Two projects $(a, A), (b, B)$ are orthogonal – written $(a, A) \perp (b, B)$ – when they have equal support and $[\![(a, A), (b, B)]\!] \neq 0, \infty$. We define the orthogonal of a set E as $E^\perp = \{(b, B) \mid \forall (a, A) \in E, (a, A) \perp (b, B)\}$ and write $E^{\perp\perp}$ the double-orthogonal $(E^\perp)^\perp$.

Orthogonality allows for a definition of types. In fact the models are defined based on two notions of types – *conducts* and *behaviours* [32]. Conducts are simple to define but while their definition is enough to define a model of multiplicative linear logic, dealing with additives requires the more refined notion of *behaviour*.

Definition 14. 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 for all (a, A) in $\mathbf{A}$ (resp. $\mathbf{A}^\perp$) and for all $\lambda \in \mathbf{R}$, $(a, A + \lambda\emptyset)$ belongs to $\mathbf{A}$ (resp. $\mathbf{A}^\perp$) as well. When both $\mathbf{A}$ and $\mathbf{A}^\perp$ are non-empty, we say $\mathbf{A}$ is *proper*.

[6] These are understood as paths whose source equals its target, quotiented by the relation $\pi \cdot e \sim e \cdot \pi$.

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:

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

However, to define additive connectives, one has to restrict the model to behaviours. 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 15 ([32, Proposition 60]). *If $\mathbf{A}$ is a proper behaviour, $(a, A) \in \mathbf{A}$ implies $a = 0$.*

Finally, let us state the fundamental theorem for the interaction graphs construction in the restricted case we just exposed[7].

Theorem 16 ([35, Theorem 1]). *For any monoid action α, the set of behaviours defines a model of Multiplicative-Additive Linear Logic (MALL) without multiplicative units.*

Most monoid actions will in fact model larger fragments of linear logic. Since we will here work in a model of Elementary Linear Logic, we now explain how exponential connectives are defined. Following our previous work [37] we fix a bijective measure-preserving pairing function: $[\cdot,\cdot] : [0,1]^2 \to [0,1]$, and define for all set A and integers $d < n$, the set $!_n^d(A)$:

$$\{(a, [x, y] \cdot \mathbf{s}, \pi) \mid (a, \mathbf{s}, \pi) \in A, nx \in [d, d+1], y \in [0,1]\}.$$

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

$$!^n_{d,d'}(f) : \left\{ \begin{array}{rcl} !^n_d(A) & \to & !^n_{d'}(B) \\ (a, x \cdot \mathbf{s}, \pi) & \mapsto & (a', y \cdot \mathbf{s}', \pi), \end{array} \right.$$

where $(a', \mathbf{s}', \pi') = f(a, \mathbf{s}, \pi)$ and $y = x + (d' - d)/n$.

This can be used to define exponential connectives by encoding the stateset into the configuration of the machine.

Definition 17. Given a graphing $G = \{(S_e^G, \phi_e^G, \mathrm{i}_\mathrm{e}^\mathrm{G} \to \mathrm{o}_\mathrm{e}^\mathrm{G}) \text{ of stateset } D = [n]$, we define the promotion $!G$ of G as the following graphing of stateset $[0]$:

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

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

This previous definition is a *perennisation* [29, 34], i.e. it maps arbitrary graphings to graphings with trivial dialect $[0]$. This implies that graphings of the form $!A$ are *duplicable* [34, Proposition 36]. As a consequence, for any conduct $!\mathbf{A}$[8] there exists a graphing C implementing contraction: $(0, C) \in !\mathbf{A} \multimap !\mathbf{A} \otimes !\mathbf{A}$.

One can also check that the above definition also allows for functorial promotion, i.e. there exists $(0, P)$ in $!(\mathbf{A} \multimap \mathbf{B}) \otimes !\mathbf{A} \multimap !\mathbf{B}$. The proof of this fact follows exactly the proof in [37].

[7] The general construction allows for other sets of weights as well as whole families of measurements [35].

[8] The conduct $!\mathbf{A}$ is not a behaviour since it cannot satisfy the inflation property. However, if $\mathbf{B}$ is an arbitrary behaviour, $!\mathbf{A} \multimap \mathbf{B}$ is a behaviour [34, Corollary 57].

Theorem 18. *Consider the monoid action* $\mathfrak{p}$ *generated by* $\bar{\mathfrak{n}}_\infty$ *together with the additional maps* $\mathtt{pair}$ *and* $\mathtt{pair}^{-1}$*, with*

$$\mathtt{pair} : (a, x \cdot y \cdot \mathbf{s}, \pi) \mapsto (a, [x, y] \cdot \mathbf{s}, \pi).$$

For any monoid action extanding $\mathfrak{p}$*, the set of conducts and behaviours is a model of Elementary Linear Logic (ELL).*

All the monoid actions considered in this paper are restrictions of the monoid action $\mathfrak{p}$. For any such action $\mathfrak{m}$, we can thus consider the model of ELL induced by $\mathfrak{p}$ and study within this model the set of $\mathfrak{m}$-graphings of type $\mathbf{Nat}_2 \multimap \mathbf{NBool}$. Before detailing this, we define the deterministic and sub-probabilistic submodels, that will be used in later sections.

2.4 Deterministic and Probabilistic Models

We furthermore use specific submodels defined by putting restrictions on the graphings considered. In order for this notion to be well-defined, one should suppose that the unit interval $[0, 1]$ endowed with multiplication is a submonoid of Ω.

Definition 19. A graphing G is *deterministic* if all edges have weight equal to 1 and the following holds:

$$\mu\left(\left\{x \in \mathbf{X} \mid \exists e, f \in E^G, e \neq f \text{ and } x \in S^G_e \cap S^G_f\right\}\right) = 0.$$

A graphing G is *sub-probabilistic* if all the edges have weight in $[0, 1]$ and the following holds:

$$\mu\left(\left\{x \in \mathbf{X} \mid \sum_{e \in E^G, x \in S^G_e} \omega^G_e > 1\right\}\right) = 0.$$

It was shown that both these notions are closed under execution [38], i.e. the execution of deterministic graphings is deterministic, similarly for sub-probabilistic graphings. As a consequence, deterministic (resp. sub-probabilistic) graphings define a submodel of $\mathbb{M}[\Omega, \mathfrak{m}]$ – the set of all Ω-weighted $\mathfrak{m}$-graphings – which we denote $\mathbb{M}^{\mathsf{det}}[\Omega, \mathfrak{m}]$ (resp. $\mathbb{M}^{\mathsf{prob}}[\Omega, \mathfrak{m}]$). Since the interpretations of proofs by graphings used in the proofs of theorem 16 and theorem 18 are all deterministic, the logic induced by these submodels is the same as the logic induced by $\mathbb{M}[\Omega, \mathfrak{m}]$.

3 CHARACTERISING COMPLEXITY CLASSES

3.1 Integers and Machines

We now review some definitions necessary to define the characterisation of complexity classes in the Interaction Graphs models [37]. We start by the representation of binary words, which is related [3, 4] to the type of binary lists in Elementary Linear Logic [11, 16]:

$$\mathrm{BList} := \forall X\, !(X \multimap X) \multimap !(X \multimap X) \multimap !(X \multimap X).$$

Intuitively a binary word, say 001, is represented as a program that takes two functions f_0 and f_1 of type $X \to X$, and produces the function $f_0 \circ f_0 \circ f_1$. This program can also be defined (in Krivine's notation) as the lambda-term $\lambda f_0 \lambda f_1 \lambda x.(f_0)(f_0)(f_1)x$. This lambda-term corresponds to a proof of BList in Elementary Linear Logic which contains exactly four axioms (fig. 2). These four axioms give rise to the representation of the proof as a graphing in the Interaction Graph model. The latter representation uses six subspaces, corresponding to the six occurrences of X in the formula BList. We will first introduce some notations for these subspaces and then define formally the graphing representations of binary words.

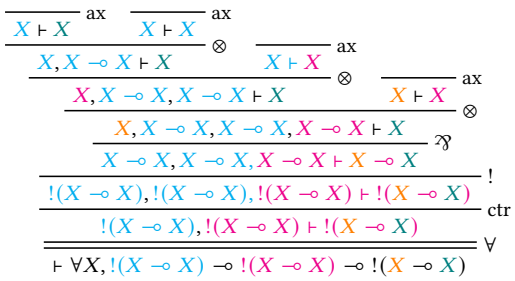


Fig. 2. Proof corresponding to $\lambda f_0.\lambda f_1.\lambda x.(f_0)(f_0)(f_1)x$.

Notation 20. We write $\Sigma^{\rightleftarrows}$ the set $\{0,1,\star\} \times \{\mathtt{in}, \mathtt{out}\}$. We also denote by $\Sigma^{\rightleftarrows}_{\mathtt{a},\mathtt{r}}$ the set $\Sigma^{\rightleftarrows} \cup \{\mathtt{a}, \mathtt{r}\}$, where $\mathtt{a}$ (resp. $\mathtt{r}$) stands for `accept` (resp. `reject`).

Initial segments of the natural numbers $\{0, 1, \dots, n\}$ are denoted $[n]$. Up to renaming, all statesets can be considered to be of this form.

Notation 21. We fix once and for all an injection Ψ from the set $\Sigma^{\rightleftarrows}_{\mathtt{a},\mathtt{r}}$ to intervals in $\mathbf{R}$ of the form $[k, k+1]$ with $k \in \mathbf{Z}$. For all $v \in \Sigma^{\rightleftarrows}_{\mathtt{a},\mathtt{r}}$, we write $\langle v \rangle^Z_Y$ the measurable subset $\Psi(v) \times Y \times Z$ of $\mathbf{X}$, where $Y \subset [0,1]^{\mathbf{N}}$ and $Z \subset \{\star, 0, 1\}^{\mathbf{N}}$.

When $Y = [0,1]^{\mathbf{N}}$ (resp. $Z = \{\star, 0, 1\}^{\mathbf{N}}$), we omit the subscript (resp. superscript). The notation extends to subsets $S \subset \Sigma^{\rightleftarrows}_{\mathtt{a},\mathtt{r}}$ by $\langle S \rangle = \cup_{v \in S} \langle v \rangle$ (a disjoint union).

Definition 22. Given a word $\mathtt{w} = a_1 a_2 \dots a_k$, we denote $[\mathtt{w}]$ the graph with set of vertices $V^{[\mathtt{w}]} \times S^{[\mathtt{w}]} = \Sigma^{\rightleftarrows} \times [k]$, set of edges $E^{[\mathtt{w}]} = \{r, l\} \times [k]$, and source and target maps $s^{[\mathtt{w}]}$ and $t^{[\mathtt{w}]}$ defined as follows:

$$\begin{array}{rcl} s^{[\mathtt{w}]} & = & (r,i) \mapsto (a_i, \mathtt{out}, \mathrm{i}) \\ & & (l,i) \mapsto (a_i, \mathtt{in}, \mathrm{i}) \\ t^{[\mathtt{w}]} & = & (r,i) \mapsto (a_{i+1}, \mathtt{in}, \mathrm{i}+1 \bmod \mathrm{k}+1) \\ & & (l,i) \mapsto (a_{i-1}, \mathtt{out}, \mathrm{i}-1 \bmod \mathrm{k}+1) \end{array}$$

Notation 23. We write $s^{[\mathtt{w}]}_{\Sigma^{\rightleftarrows}}$ (resp. $t^{[\mathtt{w}]}_{\Sigma^{\rightleftarrows}}$) the projection of the source (resp. target) map onto $\Sigma^{\rightleftarrows}$, and $s^{[\mathtt{w}]}_{[k]}$ (resp. $t^{[\mathtt{w}]}_{[k]}$) the projection of the source (resp. target) map onto $[k]$.

The graph thus defined is the discrete representations of $\mathtt{w}$, that one can relate [3] to the representation shown in fig. 2. Now, a word graphing is somehow a *geometric representation* of a graph representation of a word: it can be defined as a graphing representative with the same graph structure as a word representation.

Definition 24. Let $\mathtt{w}$ be a word $\mathtt{w} = a_1 a_2 \dots a_k$ over the alphabet Σ. The *canonical* graphing representation $\{\mathtt{w}\}$ of $\mathtt{w}$ is the graphing:

$$(\langle s^{[\mathtt{w}]}_{\Sigma^{\rightleftarrows}} \rangle, \phi_e, 1, s^{[\mathtt{w}]}_{[k]} \to t^{[\mathtt{w}]}_{[k]}) \mid (r,i) \in E^{\{\mathtt{w}\}}\},$$

where $\phi_e : \langle s^{[\mathtt{w}]}_{\Sigma^{\rightleftarrows}} \rangle \to \langle t^{[\mathtt{w}]}_{\Sigma^{\rightleftarrows}} \rangle$ is a translation. A *word graphing* W of stateset S^W is a graphing obtained from $\{\mathtt{w}\}$ by renaming the stateset w.r.t. an injection $S^W \mapsto [k]$.

We write $\mathtt{Gp}(\mathtt{w})$ the set of word graphings for $\mathtt{w}$.

Definition 25. 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 **Rep**, the set of representations of a specific word $\mathtt{w}$ is denoted $\mathbf{Rep}(\mathtt{w})$.

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

A explained in the introduction, we will be interested in machines of type $!\mathbf{Nat}_2 \multimap \mathbf{NBool}$, where **NBool** should be understood as a non-deterministic version of **Bool**.

Definition 26. We define the (unproper) behaviour **NBool** as $\mathbf{T}_{\langle \mathsf{a},\mathsf{r}\rangle}$, where for all measurable sets V the behaviour $\mathbf{T}_V$ is defined as the set of all projects of support V.

3.2 Computations, Tests and Languages

Definition 27. An $\mathfrak{m}$-graphing G is *finite* when it has a representative H whose set of edges E^H is finite.

Definition 28. For all monoid action $\mathfrak{m}$, we define $\mathbf{Pred}(\mathfrak{m})$ as the set of $\mathfrak{m}$-graphings in $!\mathbf{Nat}_2 \multimap \mathbf{NBool}$.

A *predicat* $\mathfrak{m}$*-machine* over the alphabet Σ is a finite $\mathfrak{m}$-graphing belonging to $\mathbf{Pred}(\mathfrak{m})$.

The computation of a given machine on a given input is represented by the *execution*, i.e. the computation of paths defined in Theorem 10. The result of the execution is an element of **NBool**, i.e. somehow a generalised boolean value[9].

Definition 29 (Computation)**.** Let M be a $\mathfrak{m}$-machine, w a word over the alphabet Σ and $!L \in !\mathbf{Nat}_2$. The *computation* of M over $!L$ is defined as the graphing $M :: !L \in \mathbf{NBool}$.

The principle of the approach is to use the orthogonality (which defines types) to capture the notion of acceptance. This is done using *tests*.

Definition 30. A *test* is a family of projects of support $\langle \mathsf{a}, \mathsf{r}\rangle$.

We now 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 $\mathcal{T}$:

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

The best situation is in fact when both definitions coincide, as it ensures that only one representation of w need to be considered to check whether w belongs to the language or not. This situation is captured by the notion of *uniform test*.

Definition 31 (Uniformity)**.** Let $\mathfrak{m}$ be a monoid action. 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 $\mathbf{Rep}(\mathsf{w})$:

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

We write in this case $\mathcal{L}^{\mathcal{T}}(M) = \mathcal{L}^{\mathcal{T}}_{\exists}(M) = \mathcal{L}^{\mathcal{T}}_{\forall}(M)$.

We now have introduced all the needed ingredients to state and prove characterisations of complexity classes. We will first recall previously known results [37].

Notation 32. For $U \subset \mathbf{X}$, we define Id_U as the graphing with a single edge and stateset $[0]$: $\{(\langle \mathsf{r}\rangle, \mathrm{x} \mapsto \mathrm{x}, 1 \cdot \mathbf{1}, 0 \to 0)\}$.

Proposition 33. *The test* $\mathcal{T}_{=}$*, defined as*

$$\{\mathfrak{t}^{=}_{\zeta} = (\zeta, \mathrm{Id}_{\langle \mathsf{r}\rangle}) \mid \zeta \neq 0\},$$

is uniform w.r.t. $\mathfrak{n}_\infty$*-machines.*

[9] For the specific case of "deterministic machines", the result in fact belongs to the subtype **Bool** of booleans.

Note that since it is uniform w.r.t. $\mathfrak{n}_\infty$-machines, it is uniform w.r.t. $\mathfrak{m}$-machines for any submonoid action $\mathfrak{m}$ – hence w.r.t. all monoid actions considered in this paper. We will now define classes defined from *non-deterministic* $\mathfrak{m}$-machines, i.e. finite graphing representatives of type $\mathbf{Pred}(\mathfrak{m})$ in the model $\mathbb{M}[\{0,1\},\mathfrak{m}]$ (i.e. weights are either 0 or 1).

Definition 34. We define the complexity class

$$\mathbf{Pred}^{\mathsf{co}}(\mathfrak{m}) = \{\mathcal{L}^{\mathcal{T}_-}(M) \mid M \ \mathfrak{m}\text{-machine in } \mathbb{M}[\{0,1\},\mathfrak{m}]\}.$$

We recall the main characterisation obtained in the author's previous paper [37], and refer to Theorem 40 for the definition of the characterised complexity classes. We write $co2Nfa(\mathrm{i})$ (resp. $co2Nfa(\infty)$) the set of languages decided by *non-deterministic 2-way automata* with i heads (resp. arbitrarily large number of heads) with the complementary acceptance condition: all runs are accepting. The notion is defined properly in section 5.

Theorem 35. *For all $i \in \mathbf{N}^* \cup \{\infty\}$,*

$$\mathbf{Pred}^{\mathsf{co}}(\mathfrak{m}_{\mathrm{i}}) = \textsc{co2Nfa(i)}.$$

As particular cases,

$$\mathbf{Pred}^{\mathsf{co}}(\mathfrak{m}_6) = \textit{Regular}$$
$$\mathbf{Pred}^{\mathsf{co}}(\mathfrak{m}_\infty) = \textit{coNLogspace}.$$

3.3 Characterising NLogspace

The starting point of this work was the realisation that one can define another test $\mathcal{T}_+$ capturing the notion of acceptance in NLogspace. Based on this idea, and using technical lemmas from the previous paper, we can characterise easily the hierarchy of complexity classes defined by k-head non-deterministic automata with the standard non-deterministic acceptance condition (i.e. there is at least one accepting run). We state the results and provide explanations, but we do not provide a formal proof. Indeed, while an adaptation of the techniques used in previous work [37] could be used, the result follows from the more general method exposed in the next sections.

Notation 36. For $U \subset \mathbf{X}$, we define $\mathrm{Id}_U^{1/2}$ as the graphing with a single edge and stateset $[0]$: $\{(\langle \mathsf{r}\rangle, \mathrm{x} \mapsto \mathrm{x}, \frac{1}{2}\cdot\mathbf{1}, 0 \to 0)\}$.

Proposition 37. *The test $\mathcal{T}_+$ defined as the family*

$$\{(0, \mathrm{Id}_{A_n}^{1/2}) \mid A_n = \langle \mathsf{a}\rangle_{[0,\frac{1}{\mathrm{n}}]^{\mathrm{n}}\times[0,1]^{\mathrm{N}}}, \mathrm{n} \in \mathbf{N}\}$$

is uniform w.r.t. $\mathfrak{n}_\infty$-machines.

Definition 38. We define the complexity class $\mathbf{Pred}^{\mathsf{ndet}}(\mathfrak{m})$ as the set

$$\{\mathcal{L}^{\mathcal{T}_+}(M) \mid M \ \mathfrak{m}\text{-machine in } \mathbb{M}^{\mathsf{ndet}}[\{0,1\},\mathfrak{m}]\}.$$

The considered test does indeed capture the usual condition for acceptance of non-deterministic machines. In fact, the sole element $(0, \mathrm{Id}_{\langle \mathsf{a}\rangle}^{1/2})$ is enough to obtain completeness, by a result from our earlier work [37, Proposition 46]. We do not state it here, as it is generalised by Theorem 51 below. From these results, a k-heads two-way automaton $\mathtt{M}$ accepts a word $\mathtt{w}$ if and only if there exist at least one alternating path between the graphing translation $\{\mathtt{M}\}$ of M and the word representation $!\{\mathtt{w}\}$ whose source and target is $\langle \mathsf{a}\rangle_Y$ for some subspace Y. Thus, $\mathtt{M}$ accepts $\mathtt{w}$ if and only if there are alternating cycles between $\{\mathtt{M}\} :: !\{\mathtt{w}\}$ and $\mathrm{Id}_{\langle \mathsf{a}\rangle}^{1/2}$, i.e. if and only if $[\![\{\mathtt{M}\} :: !\{\mathtt{w}\}, \mathrm{Id}_{\langle \mathsf{a}\rangle}^{1/2}]\!] \neq 0, \infty$, or equivalently if and only if $\{\mathtt{M}\} :: !\{\mathtt{w}\} \perp \mathrm{Id}_{\langle \mathsf{a}\rangle}^{1/2}$.

However, the whole family of tests is required to obtain soundness. Indeed, in the general case, it might be possible that a $\mathfrak{m}_i$-machine G passes the test $\{(0, \mathrm{Id}^{1/2}_{\langle \mathsf{a} \rangle})\}$ by taking several (possibly different) paths through the execution $G :: !\{\mathtt{w}\}$, creating a cycle of arbitrary length between $G :: !\{\mathtt{w}\}$ and $\{(0, \mathrm{Id}^{1/2}_{\langle \mathsf{a} \rangle})\}$. In that case, it is not clear that the existence of such a cycle can be decided with some automaton M. However, if $G :: !\{\mathtt{w}\}$ passes all tests in $\mathcal{T}_+$, it imposes the existence of a cycle of length 2 between $G :: !\{\mathtt{w}\}$ and $\mathrm{Id}^{1/2}_{\langle \mathsf{a} \rangle}$, something that can be decided by an automaton. The existence of the length 2 cycle is enforced by the restriction of the test to subspaces of the form $[0, \frac{1}{n}]$; we refer the the proof of lemma 58 for more details.

A simple adaptation of the arguments used in the proof of theorem 35 then provides a proof of the following. We omit the details for the moment, as the next sections will expose a generalisation of the technique (and the theorem) that also applies to the probabilistic case and to automata with a pushdown stack. Here, we write $2Nfa(\mathrm{i})$ (resp. $2Nfa(\infty)$) the set of languages decided by *non-deterministic 2-way automata* with i heads (resp. arbitrarily large number of heads) with the standard notion of acceptance: there exists an accepting run. The notion is defined properly in the next section.

Theorem 39. *For all $i \in \mathbf{N}^* \cup \{\infty\}$,*

$$\mathbf{Pred}^{\mathsf{ndet}}(\mathfrak{m}_i) = 2\mathrm{NFA}(\mathrm{i}).$$

In particular,

$$\mathbf{Pred}^{\mathsf{ndet}}(\mathfrak{m}_\infty) = \textsc{Logspace}$$

In the following sections, we will in fact establish several extensions of this result to machines with pushdown stacks.

4 STATEMENT OF THE RESULTS

4.1 Multihead automata with pushdown stacks

The proof of the characterisation theorem [37] relies on a representation of multihead automata as graphings. We here generalise the result to probabilistic automata with a pushdown stack. For practical purposes, we consider a variant of the classical notion of probabilistic two-way multihead finite automata with a pushdown stack 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 and the stack is emptied before accepting/rejecting.
- symbols from the stack are read by performing a pop instruction; if the end-of-stack symbol $\star$ is popped, it is pushed on the stack in the next transition.

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

Definition 40. A *k-heads non-deterministic two-way multihead finite automata with a pushdown stack* (2NFA+S(k)) $\mathtt{M}$ is defined as a tuple $(\Sigma, Q, \to)$, where the transition $\to$ is a relation that associates to each element of $\Sigma^k_\star \times Q$ a subset of $(\mathrm{Inst} \times Q)$ where Inst is the set of instructions: $(\{1, \ldots, k\} \times \{\mathtt{in}, \mathtt{out}\}) \times \{\mathrm{Id}, \mathsf{pop}, \mathsf{push}_1, \mathsf{push}_0, \mathsf{push}_\star\}$.

We say $\mathtt{M}$ is *deterministic* if $\to$ is a function.

Definition 41. A *k-heads probabilistic two-way multihead finite automata with a pushdown stack* (2PFA+S(k)) $\mathtt{M}$ is defined as a tuple $(\Sigma, Q, \to)$, where the transition function $\to$ is a map that associates

to each element of $\Sigma_\star^k \times Q$ a sub-probability distribution over the set $(\mathrm{Inst} \times Q)$ where Inst is the set of instructions: $(\{1,\dots,k\} \times \{\mathtt{in}, \mathtt{out}\}) \times \{\mathrm{Id}, \mathsf{pop}, \mathsf{push}_1, \mathsf{push}_0, \mathsf{push}_\star\}$.

Notation 42. The set of deterministic (resp. non-deterministic, resp. probabilistic) two-way multihead automata with k heads and *without pushdown stack* (i.e. not using stack instructions) is written $\mathbf{2dfa}(k)$ (resp. $\mathbf{2nfa}(k)$, resp. $\mathbf{2pdfa}(k)$) and the corresponding complexity class is noted 2DFA(k) (resp. 2PFA(k)). The set of all deterministic two-way multihead automata $\cup_{k\geqslant 1}\mathbf{2dfa}(k)$ is denoted by **2dfa**. We define in a similar manner the sets **2nfa** and **2pfa**. The corresponding complexity classes 2DFA(∞), 2NFA(∞), and 2PFA(∞) are known to be equal to Logspace, NLogspace, and PLogspace [19].

Notation 43. The set of k heads deterministic (resp. non-deterministic, resp. probabilistic) two-way multihead automata with k heads and a pushdown stack is written $\mathbf{2dfa + s}(k)$ (resp. $\mathbf{2nfa + s}(k)$, resp. $\mathbf{2pdfa + s}(k)$) and the corresponding complexity class is noted 2DFA+S(k) (resp. 2NFA+S(k), resp. 2PFA+S(k)). The set of all deterministic two-way multihead automata with a pushdown stack $\cup_{k\geqslant 1}\mathbf{2dfa + s}(k)$ is denoted by $\mathbf{2dfa + s}$. We define in a similar way the sets $\mathbf{2nfa + s}$ and $\mathbf{2pfa + s}$. The corresponding complexity classes 2DFA+S(∞), 2NFA+S(∞), and 2PFA+S(∞) are known to be equal to Ptime [25], Ptime, and PPtime respectively.

Remark 44. We note that non-deterministic two-way multihead automata with a pushdown stack characterise Ptime and not NPtime, as shown by Cook [9] using memoization.

4.2 Results

Before stating the theorems, we need to define the complexity classes considered in the realisability models. We already introduced two notions of tests; we will require a last one adapted to probabilistic models of computation.

Proposition 45. *Let $\eta > 0$. The test $\mathcal{T}_{+,\epsilon}$ defined by*

$$(\log(1-\frac{1}{2}.u), \mathrm{Id}^{1/2}\langle \mathsf{a}\rangle^{\mathrm{V}(\star^{\mathrm{n}})}_{[0,\frac{1}{\mathrm{n}}]^{\mathrm{n}}\times[0,1]^{\mathrm{N}}}]) \mid \mathrm{u} \in [0,\epsilon], \mathrm{n} \in \mathbf{N}\}$$

is uniform w.r.t. $\mathfrak{n}_\infty$-machines.

Definition 46. We define the complexity class $\mathbf{Pred}^{\mathsf{prob}}(\mathfrak{m})$ as the set

$$\{\mathcal{L}^{\mathcal{T}_{+,\frac{1}{2}}}(M) \mid M \ \mathfrak{m}\text{-machine in } \mathbb{M}^{\mathsf{prob}}[[0,1],\mathfrak{m}]\}.$$

In the remaining sections, we will establish the following theorem.

Theorem 47. *For all $i \in \mathbf{N}^* \cup \{\infty\}$,*

$$\mathbf{Pred}^{\mathsf{det}}(\mathfrak{m}_{\mathrm{i}}) = \mathrm{2DFA(i)} \quad \mathbf{Pred}^{\mathsf{det}}(\mathfrak{n}_{\mathrm{i}}) = \mathrm{2DFA+S(i)}$$
$$\mathbf{Pred}^{\mathsf{ndet}}(\mathfrak{m}_{\mathrm{i}}) = \mathrm{2NFA(i)} \quad \mathbf{Pred}^{\mathsf{ndet}}(\mathfrak{n}_{\mathrm{i}}) = \mathrm{2NFA+S(i)}$$
$$\mathbf{Pred}^{\mathsf{co}}(\mathfrak{m}_{\mathrm{i}}) = \mathrm{co2NFA(i)} \quad \mathbf{Pred}^{\mathsf{co}}(\mathfrak{n}_{\mathrm{i}}) = \mathrm{co2NFA+S(i)}$$
$$\mathbf{Pred}^{\mathsf{prob}}(\mathfrak{m}_{\mathrm{i}}) = \mathrm{2PFA(i)} \quad \mathbf{Pred}^{\mathsf{prob}}(\mathfrak{n}_{\mathrm{i}}) = \mathrm{2PFA+S(i)}$$

Corollary 48. *As special cases of the previous theorem,*

$$\mathbf{Pred}^{\mathsf{det}}(\mathfrak{m}_\infty) = \textit{Logspace}, \mathbf{Pred}^{\mathsf{det}}(\mathfrak{n}_\infty) = \textit{Ptime}$$
$$\mathbf{Pred}^{\mathsf{ndet}}(\mathfrak{m}_\infty) = \textit{NLogspace}, \mathbf{Pred}^{\mathsf{ndet}}(\mathfrak{n}_\infty) = \textit{Ptime}$$
$$\mathbf{Pred}^{\mathsf{co}}(\mathfrak{m}_\infty) = \textit{coNLogspace}, \mathbf{Pred}^{\mathsf{co}}(\mathfrak{n}_\infty) = \textit{Ptime}$$
$$\mathbf{Pred}^{\mathsf{prob}}(\mathfrak{m}_\infty) = \textit{PLogspace}, \mathbf{Pred}^{\mathsf{prob}}(\mathfrak{n}_\infty) = \textit{PPtime}$$

The proofs are quite similar, even though we will need to state variants of the key lemmas depending on the case considered: deterministic, non-deterministic (with different notions of acceptance), and probabilistic. However, the principle is the same and both directions rely on the fact that computation is represented by paths (cf. definition 10) and orthogonality is based on a measurement of cycles [38].

To prove completeness, we then show how any automata can be simulated by a graphing. This requires the definition of the graphing translating the automaton, and proving that the orthogonality translates the existence of accepting runs in a quantitative manner (i.e. in the case of probabilistic machines, the sum of weights of the accepting paths will be equal to the probability of accepting).

The second part of the proof, soundness, is the most involved. It requires to show that given any $\mathfrak{n}_\infty$-machine M, the computation of M on the graphing representation of a word $\mathtt{w}$ boils down to the computation of alternating paths between finite graphs (namely a graph mimicking M and the graph representation of $\mathtt{W}$).

5 COMPLETENESS

We now describe a translation of multihead automata as graphings for the set of all 2PFA+S(k).

Notation 49. For all probabilistic automaton $\mathtt{M}$ of transition $\longrightarrow$, and all pair $\mathtt{t} = ((\vec{s}, q), (i, d', q'))$, we denote by $\longrightarrow (\vec{s}, q)(i, d', q')$ the probability that $\mathtt{M}$ transitions to (i, d', q') from the configuration $(\vec{s}, q)$.

We will also write $\mathtt{t} \in\!\!\longrightarrow$ when $\longrightarrow (\vec{s}, q)(i, d', q') > 0$, i.e. when the probability that the automaton will perform the transition $\mathtt{t}$ is non-zero.

The encoding is heavy but the principle is easy to grasp. We use the stateset to keep track of the last values read by the heads, as well as the last popped symbol from the stack. The subtlety is that we also keep track of the permutation of the heads of the machine. Indeed, the graphing representation has the peculiarity that moving one head requires to use a permutation to place this head on the first copy of $[0, 1]$. As a consequence, to keep track of where the heads are positioned at a given time, we store a permutation that we update when applying each instruction. Lastly, the stack is initiated with the symbol $\star$; this is done by simply restricting the source of the edges from the initial state to the subspace $V(\star)$ of sequences starting with the symbol $\star$.

Definition 50. Let $\mathtt{M} = (\Sigma, Q, \longrightarrow)$ be a 2PFA+S(k). We define $\{\mathtt{M}\}$ a graphing in $\mathfrak{n}_\mathfrak{f}$ with dialect – set of states – $Q \times \mathfrak{G}_k \times \{\star, 0, 1\}^k \times \{\star, 0, 1\}$ as follows.

- each transition of the form $\mathtt{t} = ((\vec{s}, q), (\nu, q'))$ with $q \neq \mathtt{init}$ and $\nu = (i, d') \times \iota$ with $\iota \neq \mathsf{pop}$ gives rise to a family of edges indexed by a permutation σ and an element u of $\{\star, 0, 1\}$:

$$\begin{aligned} &\langle (a, d) \rangle \times \{(q, \sigma, \vec{s}, u)\} \\ &\quad\longrightarrow\quad \langle (s_i, d') \rangle \times \{(q', \tau_{1,\sigma(i)} \circ \sigma, \vec{s}[s_{\sigma^{-1}(1)} := s], u)\}, \end{aligned}$$

  realised by the map $\mathsf{p}_{(1,\sigma(\mathrm{i}))}$ together with the adequate map on the stack subspace and the adequate translation on $\mathbf{Z}$, and of weight $\longrightarrow (\vec{s}, q)(\nu, q')$;
- each transition of the form $\mathtt{t} = ((\vec{s}, q), (\nu, q'))$ with $q \neq \mathtt{init}$ and $\nu = (i, d') \times \mathsf{pop}$ gives rise to a family of edges indexed by a permutation σ and an element u of $\{\star, 0, 1\}$:

$$\begin{aligned} &\langle (a, d) \rangle^{V(u)} \times \{(q, \sigma, \vec{s})\} \\ &\quad\longrightarrow\quad \langle (s_i, d') \rangle \times \{(q', \tau_{1,\sigma(i)} \circ \sigma, \vec{s}[s_{\sigma^{-1}(1)} := s], u)\} \end{aligned}$$

  realised by the map $\mathsf{p}_{(1,\sigma(\mathrm{i}))}$ composed with the pop map and the adequate translation on $\mathbf{Z}$, and of weight $\longrightarrow (\vec{s}, q)(\nu, q')$;

- each transition of the form $\mathtt{t} = ((\vec{s}, q), (\nu, q'))$ with $q = \mathtt{init}$ and $\nu = (i, d') \times \iota$ with $\iota \neq \mathsf{pop}$ gives rise to a family of edges indexed by an element $v \in \{\mathsf{a}, \mathsf{r}\}$ and an element u of $\{\star, 0, 1\}$:
$$\begin{aligned}&\langle \mathsf{v}\rangle^{\mathrm{V}(\star)} \times \{(\mathrm{init}, \mathrm{Id}, \vec{\star}, \mathrm{u})\}\\ &\quad\longrightarrow\quad \langle (s_i, d')\rangle \times \{(q', \tau_{1,\sigma(i)}, \vec{s}[s_{\sigma^{-1}(1)} := s], u)\},\end{aligned}$$
realised by the map $\mathsf{p}_{(1,\sigma(\mathrm{i}))}$ together with the adequate map on the stack subspace and the adequate translation on $\mathbf{Z}$, and of weight $\to (\vec{s}, q)(\nu, q')$;
- each transition of the form $\mathtt{t} = ((\vec{s}, q), (\nu, q'))$ with $q = \mathtt{init}$ and $\nu = (i, d') \times \mathsf{pop}$ gives rise to a family of edges indexed by an element $v \in \{\mathsf{a}, \mathsf{r}\}$ and an element u of $\{\star, 0, 1\}$:
$$\begin{aligned}&\langle \mathsf{v}\rangle^{\mathrm{V}(\star)} \times \{(\mathrm{init}, \mathrm{Id}, \vec{\star}, \mathrm{u})\}\\ &\quad\longrightarrow\quad \langle (s_i, d')\rangle \times \{(q', \tau_{1,\sigma(i)}, \vec{s}[s_{\sigma^{-1}(1)} := s], u)\},\end{aligned}$$
realised by the map $\mathsf{p}_{(1,\sigma(\mathrm{i}))}$ together with the pop map on the stack subspace and the adequate translation on $\mathbf{Z}$, and of weight $\to (\vec{s}, q)(\nu, q')$.

We now state the key lemma, essential for all later results. The proof is a simple but lengthy induction.

Lemma 51. *Let* $\mathtt{M}$ *be a* $\mathbf{2pfa + s}(k)$. *Alternating paths of odd length between* $\{\mathtt{M}\}$ *and* $!\{\mathtt{w}\}$ *of source* $\langle \mathsf{a}\rangle_{\mathrm{Y}}$ *(resp.* $\langle \mathsf{r}\rangle_{\mathrm{Y}}$*) with*[10] $Y = [0, \frac{1}{\lg(\mathtt{w})}]^k \times [0,1]^{\mathbf{N}}$ *are in a weight-preserving bijective correspondence with the accepting (resp. rejecting) runs of* $\mathtt{M}$ *on input* $\mathtt{w}$.

This proposition states that given an automaton $\mathtt{M}$ and a word $\mathtt{w}$, the set of possible executions of $\mathtt{M}$ on input $\mathtt{w}$ is in bijection with the edges in $\{\mathtt{M}\} :: !\{\mathtt{w}\}$ (which, we recall, is defined as the maximal alternating paths). Moreover the probability of a given execution is equal to the weight of the corresponding edge.

Corollary 52. *The probabilistic (resp. non-deterministic) automaton* $\mathtt{M}$ *accepts* $\mathtt{w}$ *with probability* $p \in [0,1]$ *(resp. on exactly* $p \in \mathbf{N}$ *runs) if and only if* p *is equal to the sum of the weights of alternating paths between* $\{\mathtt{M}\}$ *and* $!\{\mathtt{w}\}$ *of source and target* $\langle \mathsf{a}\rangle$.

This corollary is almost enough to prove the needed completeness results. First, consider the probabilistic cas. I.e. given a probabilistic automaton $\mathtt{M}$ and a word $\mathtt{w}$:

- $\mathtt{w}$ belongs to $\mathcal{L}(\mathtt{M})$ if and only if the probability that $\mathtt{M}$ accepts on input $\mathtt{w}$ is greater than $\frac{1}{2}$;
- the probability that $\mathtt{M}$ accept on input $\mathtt{w}$ is equal to the sum of the weights of edges from $\langle \mathsf{a}\rangle$ to itself in $\{\mathtt{M}\} :: !\{\mathtt{w}\}$;
- by uniformity,
$$\mathtt{w} \in \mathcal{L}^{\mathcal{T}_{+,\frac{1}{2}}}(\{\mathtt{M}\}) \text{ iff } \{\mathtt{M}\} :: !\{\mathtt{w}\} \perp \mathcal{T}_{+,\frac{1}{2}};$$

All that is left to prove is the following lemma. Taking $\epsilon = \frac{1}{2}$, it shows that $\{\mathtt{M}\} :: !\{\mathtt{w}\} \perp \mathcal{L}^{\mathcal{T}_{+,\frac{1}{2}}}(M)$ if and only if the sum of the weights of edges from $\langle \mathsf{a}\rangle$ to itself in $\{\mathtt{M}\} :: !\{\mathtt{w}\}$ is greater than $\frac{1}{2}$. Using the three facts above, this shows that
$$\mathtt{w} \in \mathcal{L}^{\mathcal{T}_{+,\frac{1}{2}}}(\{\mathtt{M}\}) \text{ iff } \mathtt{w} \in \mathcal{L}(\mathtt{M}).$$

Lemma 53. *The sum of the weights of alternating paths between* $\{\mathtt{M}\}$ *and* $!\{\mathtt{w}\}$ *of source and target* $\langle \mathsf{a}\rangle$ *is greater than* ϵ *if and only if* $\{\mathtt{M}\} :: !\{\mathtt{w}\} \perp \mathcal{T}_{+,\epsilon}$.

[10] To understand where the subset Y comes from, we refer the reader to the proof of Lemma 56.

Proof. Let us write the weights of alternating paths between $\{\mathtt{M}\}$ and $!\{\mathtt{w}\}$ of source and target $\langle\mathtt{a}\rangle$ as $p_0, p_1, \ldots, p_k$. We use here a result from the first work on Interaction Graphs [28] showing that in the probabilistic case the measurement of two graphs $[\![G, H]\!]_m$ is equal to the measurement of the graphs $[\![\hat{G}, \hat{H}]\!]_m$ where $\hat{\ }$ fusions the edges with same source and target into a single edge by summing the weights [28, Proposition 16]. Therefore, $[\![\{\mathtt{M}\} :: !\{\mathtt{w}\}, \mathcal{T}_{+,\epsilon}]\!]$ is equal to $\epsilon - \log(1 - m(\frac{1}{2} \cdot \mathbf{1}.(\sum p_i))) = \epsilon - \log(1 - \frac{1}{2}(\sum p_i))$. Now, $\sum p_i > \epsilon$ if and only if $1 - \frac{1}{2}(\sum p_i) < 1 - \frac{1}{2}\epsilon$, if and only if $-\log(1 - \frac{1}{2}(\sum p_i)) > -\log(1 - \frac{1}{2}\epsilon)$. I.e. $\sum p_i > \epsilon$ if and only if $\log(1 - \frac{1}{2}\epsilon) - \log(1 - \frac{1}{2}(\sum p_i)) > 0$. This gives the result, i.e.

$$\sum p_i > \epsilon$$
$$\text{iff } \{\mathtt{M}\} :: !\{\mathtt{w}\} \perp (\log(1 - \frac{1}{2}.u), \mathrm{Id}^{1/2}\langle\mathtt{a}\rangle^{\mathrm{V}(\star^{\mathrm{n}})}_{[0,\frac{1}{\mathrm{n}}]^{\mathrm{n}}\times[0,1]^{\mathrm{N}}}])$$

for all $u \in [0, \epsilon]$. □

This covers the case of probabilistic automata. The deterministic and non-deterministic cases are covered in a similar way:

- $\mathtt{w}$ belongs to $\mathcal{L}(\mathtt{M})$ if and only if the number of accepting runs is greater than 1;
- the number of accepting runs of $\mathtt{M}$ on input $\mathtt{w}$ is equal to the sum of the weights of edges (i.e. the number of edges since all weights are equal to 1) from $\langle\mathtt{a}\rangle$ to itself in $\{\mathtt{M}\} :: !\{\mathtt{w}\}$;
- by uniformity,

$$\mathtt{w} \in \mathcal{L}^{\mathcal{T}_+}(\{\mathtt{M}\}) \text{ iff } \{\mathtt{M}\} :: !\{\mathtt{w}\} \perp \mathcal{T}_+.$$

The following lemma then finishes the argument for deterministic and non-deterministic machines.

Lemma 54. *There exists an alternating path between* $\{\mathtt{M}\}$ *and* $!\{\mathtt{w}\}$ *of source and target* $\langle\mathtt{a}\rangle$ *if and only if* $\{\mathtt{M}\} :: !\{\mathtt{w}\} \perp \mathcal{T}_+$.

The last case is that of non-deterministic machines with the complementary notion of acceptance. Here the key ingredients is the following lemma.

Lemma 55. *There exists an alternating path between* $\{\mathtt{M}\}$ *and* $!\{\mathtt{w}\}$ *of source and target* $\langle\mathtt{a}\rangle$ *if and only if* $\{\mathtt{M}\} :: !\{\mathtt{w}\} \not\perp \mathcal{T}_-$.

These results together provide the completeness part of theorem 47. Indeed, given any automaton $\mathtt{M}$ one can define a graphing $\{\mathtt{M}\}$ such that $\mathcal{L}(\mathtt{M}) = \mathcal{L}^{\mathcal{T}}(\{\mathtt{M}\})$ for the adequate notion of test $\mathcal{T}$. We now need to prove *soundness*, i.e. the semantic classes defined from the realisability model do not contain more languages than expected. This is more intricate, as it will be necessary to show that the behaviour of any graphing can be simulated by an automaton (while respecting the number of heads). This will be the topic of the next section.

6 SOUNDNESS

We here generalise a technical lemma from our previous paper [37, Lemma 4.14] to include probabilities and pushdown stacks. The principle is the following. Following our previous proof, the computation of a $\mathfrak{m}_i$-machine given an input w can be simulated by a computation of paths between finite graphs. This can be extended with probabilistic weights in a straightforward manner. Now, the operations on stacks could be thought of as breaking this result, since stacks are arbitrarily long. However, this can be dealt with by considering graphs whose weights are extended with an element of the monoid Θ generated by $\{0, 1, \star, c\}$ and the relations $c0 = c1 = c\star = \eta$ where η is the empty sequence, thus the neutral element of Θ. We will thus obtain that the computation of a

(Ω-weighted) $\mathfrak{n}_i$-machine given an input w can be simulated by a computation of paths between finite graphs with weights in $\Omega \times \Theta$.

Lemma 56 (Technical Lemma). *Let G be a $\mathfrak{n}_\infty$-machine. The computation of G with the representation $!\{\mathtt{w}\}$ of a word $\mathtt{w}$ is equivalent to the execution of a finite $\Omega \times \Theta$-weighted graph $\bar{G}$ and a finite graph $\bar{W}_{\mathtt{w}}$.*

Proof. The proof of this lemma follows the proof of the restricted case provided in earlier work [37]. Based on the finiteness of $\mathfrak{n}_\infty$-machines, there exist an integer N such that G is a $\mathfrak{n}_N$-machine. We now pick a word $\mathtt{w} \in \Sigma^k$. All maps realising edges in G or in $!\{\mathtt{w}\}$ are of the form $\phi \times \mathrm{Id}_{\bigtimes_{i=N+1}^{\infty}[0,1]} \times \psi$ – i.e. they are the identity on copies of $[0,1]$ indexed by natural numbers $> N$. So we can consider the underlying space to be of the form $\mathbf{Z} \times [0,1]^N \times \{\star, 0, 1\}^{\mathbf{N}}$ instead of $\mathbf{X}$ by just replacing those realisers by $\phi \times \psi$. Moreover, the maps ϕ here act either as permutations over copies of $[0,1]$ (realisers of edges of G) or as permutations over a decomposition of $[0,1]$ into k intervals (realisers of $!\{\mathtt{w}\}$). Consequently, all ϕ act as permutations over the set of N-dimension cubes of size k:

$$\{\bigtimes_{i=1}^{N} [k_i/k, (k_i+1)/k] \mid 0 \leqslant k_i \leqslant k-1\},$$

i.e. their restrictions to such N-cubes are translations.

Based on this, one can build two (thick[11]) graphs $\bar{G}$ and $\bar{W}_{\mathtt{w}}$ over the set of vertices

$$\Sigma^{\rightleftarrows} \times \{\bigtimes_{i=1}^{N} [k_i/k, (k_i+1)/k] \mid 0 \leqslant k_i \leqslant k-1\}$$

as in the proof of the restricted lemma proved in our previous work [37]. The only difference is that we will here keep track of weights and encode the stack operations as elements of Θ (we use the identification: $[[\mathsf{push}_1]] = 1$, $[[\mathsf{push}_0]] = 0$, $[[\mathsf{push}_\star]] = \star$, $[[\mathsf{pop}]] = c$):

- there is an edge in $\bar{G}$ of source $(s, (k_i)_{i=1}^N, d)$ to $(s', (k'_i)_{i=1}^N, d')$ and weight $(p, [[\psi]])$ if and only if there is an edge in G of source $\langle s\rangle \times \{d\}$ and target $\langle s'\rangle \times \{d'\}$, of weight p and whose realisation is $\phi \times \psi$ where ϕ sends 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 (of weight $(1, \epsilon)$) in $\bar{W}_{\mathtt{w}}$ 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 $\{\mathtt{w}\}$ 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 one checks that there exist an alternating path between M and $!\{\mathtt{w}\}$ of weight p and whose stack operation is equal to ψ if and only if there exist an alternating path between $\bar{M}$ and $[\mathtt{w}]$ of weight $(p, [[\psi]])$. □

Remark 57. The previous lemma is not stated in this way because space limitations did not allow us to define thick graphs and their execution. However, the graph $\bar{W}_{\mathtt{w}}$ is a refinement of the graph representation $[\mathtt{w}]$ of $\mathtt{w}$, and a more satisfying statement of the above result is that the computation of G with the representation $!\{\mathtt{w}\}$ of a word $\mathtt{w}$ is represented by the thick graph $\bar{G} ::_t \{\mathtt{w}\}$ where $::_t$ denotes the execution between thick graphs [34].

We will here work with alternating paths between the $\Omega \times \Theta$-weighted graphs. We will consider the set of Θ-trivial paths, i.e. paths whose weight is of the form (λ, c^i) (with i possibly equal to 0, with $c^0 = \eta$), and consider the sum of the weights of Θ-trivial paths as the sum in the first component, i.e. a weight in Ω.

[11] Thick graphs are graphs with, similarly to graphings, a sets of control states [34].

As a consequence of this lemma, we have that:

Lemma 58. *Let G be a $\mathfrak{n}_\infty$-machine. The computation of G with the representation $!\{\mathtt{w}\}$ of a word $\mathtt{w}$ is orthogonal to equivalent to $\mathcal{T}_{+,\epsilon}$ if and only if the sum of the weights of Θ-trivial alternating paths between $\bar{G}$ and $\bar{W}_\mathtt{w}$ from $\langle a\rangle_{\bigtimes_{i=1}^N [0,1/k]}$ to itself is greater than ϵ.*

Sketch. The orthogonality relies on alternating cycles. Here, we have that $G :: !\{\mathtt{w}\}$ is orthogonal to $\mathcal{T}_{+,\epsilon}$ if and only if the sum of the weights of alternating cycles between $G :: !\{\mathtt{w}\}$ and $\mathrm{Id}^{1/2}\langle \mathtt{a}\rangle^{\mathrm{V}(\star^\mathrm{n})}_{[0,\frac{1}{\mathrm{n}}]^\mathrm{n}\times[0,1]^\mathrm{N}}$ is of the form $a\cdot\mathbf{1}$ for $a \geqslant \epsilon$. This is because only weights of the form $a\cdot\mathbf{1}$ are considered by the map m defining the measurement. Those alternating cycles then necessarily go through at least one edge of $\mathrm{Id}^{1/2}\langle \mathtt{a}\rangle^{\mathrm{V}(\star^\mathrm{n})}_{[0,\frac{1}{\mathrm{n}}]^\mathrm{n}\times[0,1]^\mathrm{N}}$ because G and $!\{\mathtt{w}\}$ do not have weights of the form $a\cdot\mathbf{1}$. As a consequence, these cycles are concatenations of alternating paths between G and $!\{\mathtt{w}\}$ of source and target in $\langle a\rangle$ such that the overall path acts trivially on the stack.

Now, a key element in the result which has not yet appeared is that the shrinking of the support of the tests as n grows implies that the alternating paths thus concatenated need to go through the first N-cube in the finite graphs $\bar{G}$ and $\bar{W}_\mathtt{w}$. Otherwise, this concatenation of paths will not represent a cycle between $G :: !\{\mathtt{w}\}$ and $\mathrm{Id}^{1/2}\langle \mathtt{a}\rangle^{\mathrm{V}(\star^\mathrm{m})}_{[0,\frac{1}{\mathrm{m}}]^\mathrm{m}\times[0,1]^\mathrm{N}}$ for values of m larger than the length of $\mathtt{w}$. The same trick implies that the stack is emptied during the path, i.e. the weight of the path alternating between the finite graphs $\bar{G}$ and $\bar{W}_\mathtt{w}$ is required to be equal to (p, c^i).

Finally, if such cycles exists, they must be composed from Θ-trivial alternating paths between $\bar{G}$ and $\bar{W}_\mathtt{w}$ from $\langle a\rangle_{\bigtimes_{i=1}^N [0,1/k]}$ to itself. From a previous work on Interaction Graphs [28, Proposition 16] already used in the proof of lemma 53, we can fusion the edges with same source and target into a single edge by summing the weights without changing the measurement. We thus get the result. □

All that is left is to define an automaton that will compute the same language as a given $\mathfrak{n}_\infty$-machine G. Notice one subtlety here: a $\mathfrak{n}_i$ machine can use the $\mathsf{push}_\star$ instruction at any given moment. Thus the automata to be defined works with a ternary stack – over the alphabet $\{\star, 0, 1\}$ – and not a binary one. This is fine because from any automaton with a ternary stack one can define an automaton on a binary stack recognising the same language (very naively, using a representation of the ternary alphabet as words of length 2, this simply multiplies the number of states by a factor of 2).

As we have seen, for any $\mathfrak{n}_\infty$-machine G and word $\mathtt{w}$, $G :: !\{\mathtt{w}\}$ is orthogonal to $\mathcal{T}_{+,\epsilon}$ if and only if there exist a path of weight (p, c^i) with $p > \epsilon$ from the first N-dimensional cube on $\mathtt{a}$ to itself. It is then easy to define an automata $\{\mathtt{G}\}$ that computes the same language as G by simply following the transitions of $\bar{G}$, and ensuring that this automata accepts a word w with probability p if and only if there is a path of weight (p, c^i) with $p > \epsilon$ from $\langle a\rangle_{\bigtimes_{i=1}^N [0,1/k]}$ to itself. This leads to the following proposition.

Proposition 59. *Let G be a $\mathfrak{n}_i$-machine, w a word. Then $\{\mathtt{G}\}$ accepts w with probability greater than ϵ if and only if the sum of the weights in Ω of alternating paths of Θ-weight ϵ between G and $!W \otimes \mathrm{Id}_{\langle \mathtt{a}\rangle}$ from $\langle \mathtt{a}\rangle$ to itself is greater than ϵ.*

Combining the three previous statements, we obtain a proof of completeness for the probabilistic classes. The proof technique applies in a similar, yet easier, fashion to the deterministic and non-deterministic cases.

7 CONCLUSION AND PERSPECTIVES

We have shown how to extend our method based on realisability models for linear logic to capture complexity classes. We have obtained in this way numerous characterisations of numerous classes between regular languages and polynomial time, showing how the techniques applies as well to probabilistic computation. This provides the first examples of implicit characterisations of probabilistic complexity classes. This is however related to unbounded error classes, and it will be natural to try and characterise bounded-error classes. In particular, it should be possible to capture both BPL and BPP from the present work. We expect to be able to do so using the rich notion of type provided by Interaction graphs models, which allows for intersection and dependent types, as well as quantification over parameters. As an illustration, let us explain how the characterisations above can be expressed through types in the models.

We can define the language associated to a $\mathfrak{m}$-machine M and a test $\mathcal{T}$ as a type. Indeed, we say a word w is in the language defined by M if and only if $M :: \mathfrak{w} \perp \mathfrak{t}$ for all $\mathfrak{t} \in \mathcal{T}$. Using standard properties of the execution and orthogonality [35], this can be rephrased as $M :: \mathfrak{t} \perp \mathfrak{w}$. Thus, M defines a set of projects $\{M :: \mathfrak{t} \mid \mathfrak{t} \in \mathcal{T}\}$ which *tests* natural numbers, i.e. elements of $\mathbf{Nat}_2$.

Definition 60. Let M be a $\mathfrak{m}$-machine and $\mathcal{T}$ be a test. We define the type:

$$\mathbf{Lang}_{\mathcal{T}}(M) = (!\mathbf{Nat}_2^{\perp} \cup \{M :: \mathfrak{t} \mid \mathfrak{t} \in \mathcal{T}\})^{\perp}$$

In fact, $\mathbf{Lang}_{\mathcal{T}}(M)$ can also be defined as an intersection type. We write $M(\mathcal{T}) = \{M :: \mathfrak{t} \mid \mathfrak{t} \in \mathcal{T}\}$ and can obtain the following lemma.

Lemma 61. $\mathbf{Lang}_{\mathcal{T}}(M) = M(\mathcal{T})^{\perp} \cap !\mathbf{Nat}_2$.

The type represents a language in the following fashion.

Proposition 62. *Let M be a $\mathfrak{m}$-machine and $\mathcal{T}$ be a test.*

$$\mathfrak{w} \in \mathbf{Lang}_{\mathcal{T}}(M) \Leftrightarrow \exists w \in \mathcal{L}^{\mathcal{T}}(M), \mathfrak{w} \in \mathbf{Rep}(\mathtt{w})$$

Now, this is particularly interesting when one considers that the model allows for the definition of (linear) *dependent types*. Indeed, if $\mathbf{A}(\mathfrak{u})$ is a family of types (we suppose here that $\mathfrak{u}$ ranges over the type $\mathbf{U}$), the types $\sum_{\mathfrak{u}:\mathbf{U}} \mathbf{A}(u)$ and $\prod_{\mathfrak{u}:\mathbf{U}} \mathbf{A}(u)$ are well defined:

$$\sum_{\mathfrak{u}:\mathbf{U}} \mathbf{A}(u) = \{\mathfrak{u} \otimes \mathfrak{a} \mid \mathfrak{u} \in \mathbf{U}, \mathfrak{a} \in \mathbf{A}(\mathfrak{u})\}^{\perp\perp}$$

$$\prod_{\mathfrak{u}:\mathbf{U}} \mathbf{A}(u) = \{\mathfrak{f} \mid \forall \mathfrak{u} \in \mathbf{U}, \mathfrak{f} :: \mathfrak{u} \in \mathbf{A}(\mathfrak{u})\}$$

In the probabilistic model, we can use the following type to characterise[12] PPtime:

$$\sum_{M:!\mathbf{Nat}_2 \multimap \mathbf{NBool}} \mathbf{Lang}_{\mathcal{T}_{+,\frac{1}{2}}}(M).$$

Indeed, we have that:

$$\begin{aligned} &\mathbf{A} \in \text{PPtime} \\ &\Leftrightarrow \exists M : !\mathbf{Nat}_2 \multimap \mathbf{NBool}, \mathbf{A} = \mathbf{Lang}_{\mathcal{T}_{+,\frac{1}{2}}}(M). \end{aligned}$$

[12] In fact, this type is more than PPtime and the latter should be defined as a quotient to identify those M such that $\mathbf{Lang}_{\mathcal{T}}(M)$.

Noting that $\mathcal{T}_{+,\frac{1}{2}}$ can be defined as a countable intersection (thus a universal quantification), it is equal to $\forall n \in \mathbf{N}, \mathcal{T}_{+,\frac{1}{2}+\frac{1}{n}}$. The above type then becomes:

$$\sum_{M:!\mathbf{Nat}_2 \multimap \mathbf{NBool}} \forall n \in \mathbf{N},\ M(\mathcal{T}_{+,\frac{1}{2}+\frac{1}{n}})^{\perp} \cap !\mathbf{Nat}_2.$$

This type could then be used, through a quotient, to represent PPTIME in the model $\mathbb{M}^{\mathsf{prob}}[[0,1], \mathfrak{n}_\infty]$, and PLOGSPACE in the model $\mathbb{M}^{\mathsf{prob}}[[0,1], \mathfrak{m}_\infty]$.

We expect to provide types characterising bounded error predicates in the same way, providing characterisations of BPP in the model $\mathbb{M}^{\mathsf{prob}}[[0,1], \mathfrak{n}_\infty]$, and BPL in the model $\mathbb{M}^{\mathsf{prob}}[[0,1], \mathfrak{m}_\infty]$.

REFERENCES

[1] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1), 2009.

[2] Vincent Atassi, Patrick Baillot, and Kazushige Terui. Verification of ptime reducibility for system f terms via dual light affine logic. In Zoltán Ésik, editor, *Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 150–166. Springer Berlin Heidelberg, 2006. URL: http://dx.doi.org/10.1007/11874683_10, doi:10.1007/11874683_10.

[3] Clément Aubert and Thomas Seiller. Characterizing co-nl by a group action. *Mathematical Structures in Computer Science*, 26:606–638, 2016. doi:10.1017/S0960129514000267.

[4] Clément Aubert and Thomas Seiller. Logarithmic space and permutations. *Information and Computation*, 248:2–21, 2016. doi:10.1016/j.ic.2014.01.018.

[5] Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1):41 – 62, 2009. URL: http://www.sciencedirect.com/science/article/pii/S0890540108001119, doi:http://dx.doi.org/10.1016/j.ic.2008.08.005.

[6] T. Baker, J. Gill, and R. Solovay. Relativizations of the p = np question. *SIAM Journal on Computing*, 4(4):431–442, 1975. doi:10.1137/0204037.

[7] Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2, 1992. URL: https://doi.org/10.1007/BF01201998.

[8] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 CLMPS*, 1965.

[9] 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.

[10] Ugo Dal Lago and Paolo Parisen Toldin. A higher-order characterization of probabilistic polynomial time. *Information and Computation*, 241:114–141, 2015.

[11] 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.

[12] Jack Edmonds. Paths, trees and flowers. *Canad. J. Math.*, 17:449–467, 1965.

[13] Damien Gaboriau. Coût des relations d'équivalence et des groupes. *Inventiones Mathematicae*, 139:41–98, 2000. doi:10.1007/s002229900019.

[14] Damien Gaboriau. Invariants ℓ^2 de relations d'équivalence et de groupes. *Publ. Math. Inst. Hautes Études Sci*, 95(93-150):15–28, 2002.

[15] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987. doi:10.1016/0304-3975(87)90045-4.

[16] Jean-Yves Girard. Light linear logic. In *Selected Papers from the International Workshop on Logical and Computational Complexity*, LCC '94, pages 145–176, London, UK, UK, 1995. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=648045.745202.

[17] 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.

[18] Juris Hartmanis and Richard Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117, 1965.

[19] Markus Holzer, Martin Kutrib, and Andreas Malcher. Complexity of multi-head finite automata: Origins and directions. *Theoretical Computer Science*, 412(1):83 – 96, 2011. Complexity of Simple Programs. URL: http://www.sciencedirect.com/science/article/pii/S0304397510004597, doi:https://doi.org/10.1016/j.tcs.2010.08.024.

[20] Reinhard Kahle and Isabel Oitavem. Towards recursion schemata for the probabilistic class pp. *Lógica e Computação*, page 91.

[21] Laszlo Kalmar. Egyszerii pelda eldonthetetlen aritmetikai problemara. *Matematikai es Fizikai Lapok*, (50):1–23, 1943.

[22] Yves Lafont. Soft linear logic and polynomial time. *Theor. Comput. Sci.*, 318(1-2):163–180, June 2004. URL: http://dx.doi.org/10.1016/j.tcs.2003.10.018, doi:10.1016/j.tcs.2003.10.018.
[23] D. Leivant and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundam. Inform.*, 19, 1993.
[24] Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: Substitution and poly-space. *Lecture Notes in Computer Science*, 933, 1994. doi:10.1007/BFb0022277.
[25] Ioan I. Macarie. Multihead two-way probabilistic finite automata. *Theory Comput. Syst.*, 30(1):91–109, 1997. URL: http://dx.doi.org/10.1007/s002240000043, doi:10.1007/s002240000043.
[26] A. A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55, 1997. doi:10.1006/jcss.1997.1494.
[27] Thomas Seiller. Interaction graphs and complexity I. In preparation.
[28] Thomas Seiller. Interaction graphs: Multiplicatives. *Annals of Pure and Applied Logic*, 163:1808–1837, December 2012. doi:10.1016/j.apal.2012.04.005.
[29] Thomas Seiller. *Logique dans le facteur hyperfini : géometrie de l'interaction et complexité*. PhD thesis, Université Aix-Marseille, 2012. URL: http://tel.archives-ouvertes.fr/tel-00768403/.
[30] Thomas Seiller. Measurable preorders and complexity. Topology, Algebra and Categories in Logic Conference, 2015.
[31] Thomas Seiller. Towards a *Complexity-through-Realizability* theory. http://arxiv.org/pdf/1502.01257, 2015.
[32] Thomas Seiller. Interaction graphs: Additives. *Annals of Pure and Applied Logic*, 167:95 – 154, 2016. doi:10.1016/j.apal.2015.10.001.
[33] Thomas Seiller. Interaction graphs: Full linear logic. In *IEEE/ACM Logic in Computer Science (LICS)*, 2016. URL: http://arxiv.org/pdf/1504.04152.
[34] Thomas Seiller. Interaction graphs: Exponentials. *Logical Methods in Computer Science*, 2017. Under revision. URL: http://arxiv.org/pdf/1312.1094.
[35] Thomas Seiller. Interaction graphs: Graphings. *Annals of Pure and Applied Logic*, 168(2):278–320, 2017. doi:10.1016/j.apal.2016.10.007.
[36] Thomas Seiller. A correspondence between maximal abelian sub-algebras and linear logic fragments. *Mathematical Structures in Computer Science*, 28(1):77–139, 2018. doi:10.1017/S0960129516000062.
[37] Thomas Seiller. Interaction graphs: Nondeterministic automata. *ACM Transaction in Computational Logic*, 19(3), 2018.
[38] Thomas Seiller. Zeta functions and the (linear) logic of markov processes. submitted, https://hal.archives-ouvertes.fr/hal-02458330, 2022.
[39] Thomas Seiller, Luc Pellissier, and Ulysse Léchine. On the power of euclidean division: lower bounds for algebraic machines, semantically. submitted, 2022.