Disclaimer

This document is a post-print version of an article published in the Lecture Notes in Computer Science series (as allowed by their policy).

To cite the published version (which is the only worth citing), please use

```
@incollection{Aubert2014Unification,
author={Aubert, Clément and Bagnol, Marc},
title={Unification and Logarithmic Space},
year={2014},
isbn={978-3-319-08917-1},
booktitle={Rewriting and Typed Lambda Calculi},
volume={8560},
series={Lecture Notes in Computer Science},
editor={Dowek, Gilles},
doi={10.1007/978-3-319-08918-8_6},
publisher={Springer International Publishing},
pages={77--92},
language={English},
note={Post-print version available at
     \url{http://aubert.perso.math.cnrs.fr/recherche/unification-and-logarithmic-space.pdf}}
}
```

2017/03/15

Unification and Logarithmic Space

Clément Aubert and Marc Bagnol*

Aix-Marseille Université, CNRS, I2M, UMR 7373, 13453 Marseille, France

Abstract. We present an algebraic characterization of the complexity classes Logspace and NLogspace, using an algebra with a composition law based on unification. This new bridge between unification and complexity classes is inspired from proof theory and more specifically linear logic and Geometry of Interaction.
We show how unification can be used to build a model of computation by means of specific subalgebras associated to finite permutations groups.
We then prove that whether an observation (the algebraic counterpart of a program) accepts a word can be decided within logarithmic space. We also show that the construction can naturally represent pointer machines, an intuitive way of understanding logarithmic space computing.

Keywords: Implicit Complexity, Unification, Logarithmic Space, Proof Theory, Pointer Machines, Geometry of Interaction.

Introduction

Proof theory and complexity theory. There is a longstanding tradition of relating proof theory (more specifically linear logic [1]) and implicit complexity theory that dates back to the introduction of bounded [2] and light [3] logics. Control over the modalities [4,5], type assignment [6] and stratification of exponential boxes [7], to name a few, led to a clearer understanding of the complexity bounds linear logic could entail on the cut-elimination procedure.

We propose to push further this approach by adopting a more semantical and algebraic point of view that will allow us to capture non-deterministic logarithmic space computation.

Geometry of Interaction. As the study of cut-elimination has grown as a central topic in proof theory, its mathematical modeling became of great interest. The Geometry of Interaction [8] research program led to mathematical models of cut-elimination in terms of paths in proofnets [9], token machines [10] and operator algebras [11]. It was already used with complexity concerns [12,13].

Recent works [13,14,15] studied the link between Geometry of Interaction and logarithmic space, relying on the theory of von Neumann algebras. Those three articles are indubitably sources of inspiration of this work, but the whole construction is made anew, in a simpler framework.

* This work was partly supported by the ANR-10-BLAN-0213 Logoi and the ANR-11-BS02-0010 Récré.

Unification. Unification is one of the key-concepts of theoretical computer science, for it is used in logic programming and is a classical subject of study for complexity theory. It was shown [16,17] that one can model cut-elimination with unification techniques.

Execution will be expressed in terms of matching in a *unification algebra*. This is a simple framework, yet expressive enough to encode the action of finite permutation groups on an unbounded tensor product, which is a crucial ingredient of our construction.

Contribution. We carry on the methodology of bridging Geometry of Interaction and complexity theory with a renewed approach. It relies on an simpler representation of execution in a unification-based algebra, proved to capture exactly logarithmic space complexity.

While the representation of inputs (words over a finite alphabet) comes from the classical Church representation of lists, observations (the algebraic counterpart of programs) are shown to correspond very naturally to a notion of pointer machines. This correspondence allows us to prove that reversibility (of machines) is related to the algebraic notion of isometricity (of observations).

Organization of this article. In Sect.1 we review some classical results on unification of first-order terms and use them to build the algebra that will constitute our computational setting.

We explain in Sect.2 how words and computing devices (observations) can be modeled by particular elements of this algebra. The way they interact to yield a notion of language recognized by an observation is described in Sect.3.

Finally, we show in Sect.4 that our construction captures exactly logarithmic space computation, both deterministic and non-deterministic.

1 The Unification Algebra

1.1 Unification

Unification can be generally thought of as the study of formal solving of equations between terms.

This topic was introduced by Herbrand, but became really widespread after the work of J. A. Robinson on automated theorem proving. The unification technique is also at the core of the logic programming language Prolog and type inference for functional programming languages such as CaML and Haskell.

Specifically, we will be interested in the following problem:

Given two terms, can they be "made equal" by replacing their variables?

Definition 1 ***(terms)***
We consider the following set of first-order terms

$$\mathtt{T} ::= \quad x, y, z, \ \ldots \ \mid \mathtt{a}, \mathtt{b}, \mathtt{c}, \ \ldots \ \mid \mathtt{T} \bullet \mathtt{T}$$

where $x, y, z, \cdots \in \mathtt{V}$ *are variables,* $\mathtt{a}, \mathtt{b}, \mathtt{c}, \ \ldots$ *are constants and* $\bullet$ *is a binary function symbol.*

For any $t \in \mathtt{T}$, *we will write* $\mathtt{Var}(t)$ *the set of variables occurring in* t. *We say that a term is closed when* $\mathtt{Var}(t) = \varnothing$, *and denote* $\mathtt{T_c}$ *the set of closed terms.*

Notation. *The binary function symbol* $\bullet$ *is not associative, but we will write it by convention as* right associating *to lighten notations:* $t \bullet u \bullet v := t \bullet (u \bullet v)$

Definition 2 ***(substitution)***
A substitution *is a map* $\theta : \mathtt{V} \to \mathtt{T}$ *such that the set* $\mathtt{Dom}(\theta) := \{ v \in \mathtt{V} \,|\, \theta(v) \neq v \}$ *(the* domain *of* θ*) is finite. A substitution with domain* $\{ x_1, \ldots, x_n \}$ *such that* $\theta(x_1) = u_1$, $\ldots$, $\theta(x_n) = u_n$ *will be written as* $\{ x_1 \mapsto u_1 \,;\, \ldots \,;\, x_n \mapsto u_n \}$.

If $t \in \mathtt{T}$ *is a term we write* $t.\theta$ *the term* t *where any occurrence of any variable* x *has been replaced by* $\theta(x)$.

If $\theta = \{ x_i \mapsto u_i \}$ *and* $\psi = \{ y_j \mapsto v_j \}$, *their* composition *is defined as*

$$\theta ; \psi \;:=\; \{ x_i \mapsto u_i.\psi \} \,\cup\, \{ y_j \mapsto v_j \mid y_j \notin \mathtt{Dom}(\theta) \}$$

Remark. *The composition of substitutions is such that* $t.(\theta ; \psi) = (t.\theta).\psi$ *holds.*

Definition 3 ***(renamings and instances)***
A renaming *is a substitution* α *such that* $\alpha(\mathtt{V}) \subseteq \mathtt{V}$ *and that is bijective. A term* t' *is a* renaming *of* t *if* $t' = t.\alpha$ *for some renaming* α.

Two substitutions θ, ψ *are equal* up to renaming *if there is a renaming* α *such that* $\psi = \theta ; \alpha$.

A substitution ψ *is an* instance *of* θ *if there is a substitution* σ *such that* $\psi = \theta ; \sigma$.

Proposition 4
Let θ, ψ *be two substitutions. If* θ *is an instance of* ψ *and* ψ *is an instance of* θ, *then they are equal up to renaming.*

Definition 5 ***(unification)***
Two terms t, u *are* unifiable *if there is a substitution* θ *such that* $t.\theta = u.\theta$.

We say that θ *is a* most general unifier (MGU) *of* t, u *if any other unifier of* t, u *is an instance of* θ.

Remark. *It follows from Proposition 4 that any two MGU of a pair of terms are equal up to renaming.*

We will be interested mostly in the weaker variant of unification where one can first perform renamings on terms so that their variables are distinct, we introduce therefore a specific vocabulary for it.

Definition 6 ***(disjointness and matching)***
Two terms t, u *are* matchable *if* t', u' *are unifiable, where* t', u' *are renamings (Definition 3) of* t, u *such that* $\mathtt{Var}(t') \cap \mathtt{Var}(u') = \varnothing$.
If two terms are not matchable, they are said to be disjoint.

Example. *x and $\mathtt{f}_\bullet x$ are not unifiable.*
But they are matchable, as $x.\{\, x \mapsto y\,;\, y \mapsto x\,\} = y$ which is unifiable with $\mathtt{f}_\bullet x$.
More generally, disjointness is stronger than non-unifiability.

The crucial feature of first-order unification is the (decidable) existence of most general unifiers for unification problems that have a solution.

Proposition 7 ***(MGU)***
If a unification problem has a unifier, then it has a MGU.
Whether two terms are unifiable and, in case they are, finding a MGU is decidable.

As unification grew in importance, the study of its complexity gained in attention. A complete survey [18] tells the story of the bounds getting sharpened: general first-order unification was finally proved [19] to be a PTIME-complete problem.

In this article, we are concerned with a very much simpler case of the problem: the matching (Definition 6) of linear terms (*ie.* where variables occur at most once). This case can be solved in a space-efficient way.

Proposition 8 ***(matching in logarithmic space [20, Lemma 20])***
Whether two linear terms t, u with disjoint sets of variables are unifiable, and if so finding a MGU, can be computed in logarithmic space in the size[1] *of t, u on a deterministic Turing machine*

The lemma in [20] actually states that the problem is in NC^1, a complexity class of parallel computation known to be included in LOGSPACE.
We will use only a special case of the result, matching a linear term against a closed term.

1.2 Flows and Wirings

We now use the notions we just saw to build an algebra with a product based on unification. Let us start with a monoid with a partially defined product, which will be the basis of the construction.

Definition 9 ***(flows)***
A flow *is an oriented pair written $t \leftarrow u$ with $t, u \in \mathtt{T}$ such that $\mathtt{Var}(t) = \mathtt{Var}(u)$.*
Flows are considered up to renaming: for any renaming α, $t \leftarrow u \;=\; t.\alpha \leftarrow u.\alpha$.
We will write $\mathcal{F}$ the set of (equivalence classes of) flows.
We set $I := x \leftarrow x$ and $(t \leftarrow u)^\dagger := u \leftarrow t$ so that $(.)^\dagger$ is an involution of $\mathcal{F}$.

A flow $t \leftarrow u$ can be thought of as a `'match ... with u -> t'` in a ML-style language. The composition of flows follows this intuition.

Definition 10 ***(product of flows)***
Let $u \leftarrow v \in \mathcal{F}$ and $t \leftarrow w \in \mathcal{F}$. Suppose we have chosen two representatives of the renaming classes such that their sets of variables are disjoint.
The product *of $u \leftarrow v$ and $t \leftarrow w$ is defined if v, t are unifiable with MGU θ (the choice of a MGU does not matter because of the remark following Definition 5) and in that case: $(u \leftarrow v)(t \leftarrow w) \;:=\; u.\theta \leftarrow w.\theta$.*

[1] The *size* of a term is the total number of occurrences of symbols in it.

Definition 11 ***(action on closed terms)***
If $t \in \mathtt{T_c}$ *is a closed term,* $(u \leftharpoonup v)(t)$ *is defined whenever* t *and* v *are unifiable, with MGU* θ*, in that case* $(u \leftharpoonup v)(t) := u.\theta$

Examples. *Composition of flows:* $(x\bullet\mathtt{c} \leftharpoonup (\mathtt{c}\bullet\mathtt{c})\bullet x)(y\bullet z \leftharpoonup z\bullet y) = x\bullet\mathtt{c} \leftharpoonup x\bullet\mathtt{c}\bullet\mathtt{c}$.
Action on a closed term: $(x\bullet\mathtt{c} \leftharpoonup x\bullet\mathtt{c}\bullet\mathtt{c})(\mathtt{d}\bullet\mathtt{c}\bullet\mathtt{c}) = \mathtt{d}\bullet\mathtt{c}$.
Remark. *The condition on variables ensures that the result is a closed term (because* $\mathtt{Var}(u) \subseteq \mathtt{Var}(v)$*) and that the action is injective on its domain of definition (because* $\mathtt{Var}(v) \subseteq \mathtt{Var}(u)$*). Moreover, the action is compatible with the product of flows:* $l(k(t)) = (l\,k)(t)$ *and both are defined at the same time.*

By adding a formal element $\perp$ (representing the failure of unification) to the set of flows, one could turn the product into a completely defined operation, making $\mathcal{F}$ an *inverse monoid*. However, we will need to consider the wider algebra of *sums* of flows that is easily defined directly from the partially defined product.

Definition 12 ***(wirings)***
Wirings *are* $\mathbb{C}$*-linear combinations of elements of* $\mathcal{F}$ *(formally: almost-everywhere null functions from* $\mathcal{F}$ *to* $\mathbb{C}$*), endowed with the following operations:*

$$\left(\sum_i \lambda_i\, l_i\right)\left(\sum_j \mu_j\, k_j\right) := \sum_{\substack{i,j \text{ such that} \\ (l_i k_j) \text{ is defined}}} \lambda_i\mu_j(l_i\, k_j) \qquad \textit{(with } \lambda_i,\mu_j \in \mathbb{C} \textit{ and } l_i,k_j \in \mathcal{F}\textit{)}$$

$$\textit{and} \qquad \left(\sum_i \lambda_i\, l_i\right)^\dagger := \sum_i \overline{\lambda}_i\, l_i^\dagger \qquad \textit{(where } \overline{\lambda} \textit{ is the complex conjugate of } \lambda\textit{)}$$

We write $\mathcal{U}$ *the set of wirings and refer to it as the* unification algebra.

Remark. *Indeed,* $\mathcal{U}$ *is a unital* $*$*-algebra: it is a* $\mathbb{C}$*-algebra (considering the product defined above) with an involution* $(.)^\dagger$ *and a unit* I.

Definition 13 ***(partial isometries)***
A partial isometry *is a wiring* $U \in \mathcal{U}$ *satisfying* $UU^\dagger U = U$.

Example. $(\mathtt{c}\bullet x \leftharpoonup x\bullet\mathtt{d}) + (\mathtt{d}\bullet\mathtt{c} \leftharpoonup \mathtt{c}\bullet\mathtt{c})$ *is a partial isometry.*

While $\mathcal{U}$ offers the general algebraic background to work in, we will need to consider particular kind of wirings to study computation.

Definition 14 ***(concrete and isometric wirings)***
A wiring is concrete *whenever it is a sum of flows with all coefficients equal to* 1.
An isometric wiring *is a concrete wiring that is also a partial isometry.*
Given a set of wirings E *we write* E^+ *for the set of all concrete wirings of* E.

Isometric wirings enjoy a direct characterization.

Proposition 15 ***(isometric wirings)***
The isometric wirings are exactly the wirings of the form $\sum_i u_i \leftharpoonup t_i$ *with the* u_i *pairwise disjoint (Definition 6) and* t_i *pairwise disjoint.*

It will be useful to consider the action of wirings on closed terms. For this purpose we extend Definition 11 to wirings.

Definition 16 ***(action on closed terms)***
Let $\mathbb{V}_{\mathtt{c}}$ *be the free* $\mathbb{C}$*-vector space over* $\mathtt{T}_{\mathtt{c}}$ *.*
Wirings act on base vectors of $\mathbb{V}_{\mathtt{c}}$ *in the following way*

$$\left(\sum_i \lambda_i\, l_i\right)(t) := \sum_{\substack{i \text{ such that} \\ l_i(t) \text{ is defined}}} \lambda_i\big(l_i(t)\big) \quad \in \mathbb{V}_{\mathtt{c}}$$

which extends by linearity into an action on the whole $\mathbb{V}_{\mathtt{c}}$ *.*

Isometric wirings have a particular behavior in terms of this action.

Lemma 17 ***(isometric action)***
Let F *be an isometric wiring and* t *a closed term. We have that* $F(t)$ *and* $F^\dagger(t)$ *are either* 0 *or another closed term* t' *(seen as an element of* $\mathbb{V}_{\mathtt{c}}$*).*

1.3 Tensor Product and Permutations

We define now the representation in $\mathcal{U}$ of structures that provide enough expressivity to model computation.
Unbounded tensor products will allow to represent data of arbitrary size, and finite-support permutations will be used to manipulate these data.

Notations. *Given any set of wirings or closed terms* E *, we write* $\mathtt{Vect}(E)$ *the vector space generated by* E *,* ie. *the set of* finite *linear combinations of elements of* E *(for instance* $\mathtt{Vect}(\mathtt{T}_{\mathtt{c}}) = \mathbb{V}_{\mathtt{c}}$*).*
Moreover, we set $\mathcal{I} := \{\, \lambda I \mid \lambda \in \mathbb{C} \,\}$ *(with* $I = x \leftharpoondown x$ *as in Definition 9) which is the* $*$*-algebra of multiples of the identity, and* $u \leftrightharpoons v := u \leftharpoondown v + v \leftharpoondown u$ *.*
For brevity we write "$*$*-algebra" instead of the more correct "*$*$*-subalgebra of* $\mathcal{U}$*"* (ie. *a subset of* $\mathcal{U}$ *that is stable by linear combinations, product and* $(.)^\dagger$*).*

Definition 18 ***(tensor product)***
Let $u \leftharpoondown v$ *and* $t \leftharpoondown w$ *be two flows. Suppose we have chosen representatives of these renaming classes that have their sets of variables disjoint. We define their* tensor product *as* $(u \leftharpoondown v) \mathbin{\dot\otimes} (t \leftharpoondown w) := u \bullet t \leftharpoondown v \bullet w$ *. The operation is extended to wirings by bilinearity.*
Given two $*$*-algebras* $\mathcal{A}, \mathcal{B}$ *, we define their tensor product as the* $*$*-algebra*

$$\mathcal{A} \mathbin{\dot\otimes} \mathcal{B} := \mathtt{Vect}\left\{\, F \mathbin{\dot\otimes} G \;\middle|\; F \in \mathcal{A},\, G \in \mathcal{B} \,\right\}$$

This actually defines an embedding of the algebraic tensor product of $*$-algebras into $\mathcal{U}$, which means in particular that $(F \mathbin{\dot\otimes} G)(P \mathbin{\dot\otimes} Q) = (FP) \mathbin{\dot\otimes} (GQ)$. It ensures also that the $\dot\otimes$ operation indeed yields $*$-algebras.

Notation. *As* $\bullet$*, the* $\dot\otimes$ *operation is not associative. We carry on our convention and write it as* right associating*:* $\mathcal{A} \mathbin{\dot\otimes} \mathcal{B} \mathbin{\dot\otimes} \mathcal{C} := \mathcal{A} \mathbin{\dot\otimes} (\mathcal{B} \mathbin{\dot\otimes} \mathcal{C})$ *.*

Definition 19 ***(unbounded tensor)***
Let $\mathcal{A}$ *be a* $*$*-algebra. We define the* $*$*-algebras* $\mathcal{A}^{\otimes n}$ *for all* $n \in \mathbb{N}$ *as*

$$\mathcal{A}^{\otimes 0} := \mathcal{I} \quad \text{and} \quad \mathcal{A}^{\otimes n+1} := \mathcal{A} \mathbin{\dot\otimes} \mathcal{A}^{\otimes n}$$

and the $*$*-algebra* $\mathcal{A}^{\otimes\infty} := \bigcup_{n \in \mathbb{N}} \mathcal{A}^{\otimes n}$ *.*

We will consider finite permutations, but allow them to be composed even when their domain of definition do not match.

Notations. *Let* $\mathfrak{S}_n$ *be the set of finite permutations over* $\{1,\dots,n\}$*, if* $\sigma \in \mathfrak{S}_n$*, we define* $\sigma_{+k} \in \mathfrak{S}_{n+k}$ *as the permutation* σ *extended to* $\{1,\dots,n,\dots,n+k\}$ *letting* $\sigma_{+k}(n+i) := n+i$ *for* $i \in \{1,\dots,k\}$*.*
We also write $I_k := Id_{\{1,\dots,k\}} \in \mathfrak{S}_k$*.*

Definition 20 ***(representation)***
To a permutation $\sigma \in \mathfrak{S}_n$ *we associate the flow*

$$[\sigma] := x_1 \bullet x_2 \bullet \cdots \bullet x_n \bullet y \leftharpoonup x_{\sigma(1)} \bullet x_{\sigma(2)} \bullet \cdots \bullet x_{\sigma(n)} \bullet y$$

A permutation $\sigma \in \mathfrak{S}_n$ will act on the first n components of the unbounded tensor product (Definition 19) by swapping them and leaving the rest unchanged. The wirings $[\sigma]$ internalize this action: in the above definition, the variable y at the end stands for the components that are not affected.

Example. *Let* $\tau \in \mathfrak{S}_2$ *be the permutation swapping the two elements of* $\{1,2\}$ *and* $U_1 \dot{\otimes} U_2 \dot{\otimes} U_3 \dot{\otimes} I \in \mathcal{U}^{\otimes 3} \subseteq \mathcal{U}^{\otimes\infty}$*. We have* $[\tau] = x_1 \bullet x_2 \bullet y \leftharpoonup x_2 \bullet x_1 \bullet y$ *and* $[\tau](U_1 \dot{\otimes} U_2 \dot{\otimes} U_3 \dot{\otimes} I)[\tau]^\dagger = U_2 \dot{\otimes} U_1 \dot{\otimes} U_3 \dot{\otimes} I$*.*

Proposition 21 ***(representation)***
For $\sigma \in \mathfrak{S}_n$ *and* $\tau \in \mathfrak{S}_{n+k}$ *we have*

$$[\sigma_{+k}] = [\sigma][I_{n+k}] = [I_{n+k}][\sigma] \qquad [\sigma_{+k} \circ \tau] = [\sigma][\tau] \qquad \text{and} \qquad [\sigma^{-1}] = [\sigma]^\dagger$$

Definition 22 ***(permutation algebra)***
For $n \in \mathbb{N}$ *we set* $[\mathfrak{S}_n] := \{\, [\sigma] \mid \sigma \in \mathfrak{S}_n \,\}$ *and* $\mathcal{S}_n := \mathtt{Vect}[\mathfrak{S}_n]$*.*
We define then $\mathcal{S} := \bigcup_{n\in\mathbb{N}} \mathcal{S}_n$*, which we call the* permutation algebra*.*

Proposition 21 ensures that the $\mathcal{S}_n$ and $\mathcal{S}$ are $*$-algebras.

2 Words and Observations

The representation of words over an alphabet in the unification algebra directly comes from the translation of Church lists in linear logic and their interpretation in Geometry of Interaction models [11,16].
This proof-theoretic origin is an useful guide for intuition, even if we give here a more straightforward definition of the notion.

From now on, we fix a set of two distinguished constant symbols $\mathtt{LR} := \{\, \mathtt{L}, \mathtt{R} \,\}$.

Definition 23 ***(word algebra)***
To a set S *of closed terms, we associate the* $*$*-algebra*

$$S^* := \mathtt{Vect}\,\{\, t \leftharpoonup u \mid t,u \in S \,\}$$

(which is indeed an algebra because unification of closed terms is simply equality)

The word algebra *associated to a finite set of constant symbols* Σ *is the* $*$*-algebra defined as*

$$\mathcal{W}_\Sigma := (\mathcal{I} \dot{\otimes} \Sigma^* \dot{\otimes} \mathtt{LR}^*) \dot{\otimes} (\mathtt{T}_\mathtt{c}^*)^{\otimes 1}$$

($\mathtt{T}_\mathtt{c}$ *is the set of all closed terms,* $\mathcal{I}$ *is defined at the beginning of Sect.1.3*
$\dot{\otimes}$ *is as in Definition 18 and* $(.)^{\otimes 1}$ *is the case* $n = 1$ *of Definition 19)*

The words we consider are cyclic, with a begin/end marker $\star$, a reserved constant symbol. For example the word $\mathtt{0010}$ is to be thought of as $\star\mathtt{0010} = \mathtt{10}\star\mathtt{00} = \mathtt{0}\star\mathtt{001} = \cdots$.
We consider therefore that the alphabet Σ always contains the symbol $\star$.

Definition 24 ***(word representation)***
Let $W = \star\mathtt{c}_1 \dots \mathtt{c}_n$ *be a word over* Σ *and* $t_0, t_1, \dots, t_n$ *be distinct closed terms. The* representation $W(t_0, t_1, \dots, t_n) \in \mathcal{W}_\Sigma^+$ *with respect to* $t_0, t_1, \dots, t_n$ *of* W *is an isometric wiring (Definition 14), defined as*

$$\begin{aligned} W(t_0, t_1, \dots, t_n) := \quad & x \bullet \star \bullet \mathtt{R} \bullet (t_0 \bullet y) \leftrightharpoons x \bullet \mathtt{c}_1 \bullet \mathtt{L} \bullet (t_1 \bullet y) \\ + \; & x \bullet \mathtt{c}_1 \bullet \mathtt{R} \bullet (t_1 \bullet y) \leftrightharpoons x \bullet \mathtt{c}_2 \bullet \mathtt{L} \bullet (t_2 \bullet y) \\ + \; & \cdots \\ + \; & x \bullet \mathtt{c}_n \bullet \mathtt{R} \bullet (t_n \bullet y) \leftrightharpoons x \bullet \star \bullet \mathtt{L} \bullet (t_0 \bullet y) \end{aligned}$$

We now define *observations*, programs computing on representations of words. They lie in a particular $*$-algebra based on the representation of permutations presented in Sect.1.3.

Definition 25 ***(observation algebra)***
An observation *over a finite set of symbols* Σ *is any element of* $\mathcal{O}_\Sigma^+$ *where* $\mathcal{O}_\Sigma := (\mathtt{T}_\mathtt{c}^* \dot{\otimes} \Sigma^* \dot{\otimes} \mathtt{LR}^*) \dot{\otimes} \mathcal{S}$*, i.e. a* finite *sum of flows of the form*

$$(s' \bullet \mathtt{c}' \bullet \mathtt{d}' \leftharpoondown s \bullet \mathtt{c} \bullet \mathtt{d}) \dot{\otimes} [\sigma]$$

with s, s' *closed terms,* $\mathtt{c}, \mathtt{c}' \in \Sigma$*,* $\mathtt{d}, \mathtt{d}' \in \mathtt{LR}$ *and* σ *is a permutation.*

Moreover when an observation happens to be an isometric wiring, we will call it an isometric observation*.*

3 Normativity: Independence from Representations

We are going to define how observations accept and reject words. This needs to be discussed, because there is an issue with word representations: an observation is an element of $\mathcal{U}$ and can therefore only interact with *representations* of a word, and there are many possible representation of the same word (in Definition 24, different choices of closed terms lead to different representations). Therefore one has to ensure that acceptance or rejection is independent of the representation, so that the notion makes the intended sense.
The termination of computations will correspond to the algebraic notion of *nilpotency*, which we recall here.

Definition 26 ***(nilpotency)***
A wiring F *is* nilpotent *if* $F^n = 0$ *for some* n.

Definition 27 ***(automorphism)***
An automorphism *of a* $*$*-algebra* $\mathcal{A}$ *is a linear application* $\varphi : \mathcal{A} \to \mathcal{A}$ *such that for all* $F, G \in \mathcal{A}$*:* $\varphi(FG) = \varphi(F)\varphi(G)$, $\varphi(F^\dagger) = \varphi(F)^\dagger$ *and* φ *is injective*.

Example. $\varphi(U_1 \mathbin{\dot\otimes} U_2) := U_2 \mathbin{\dot\otimes} U_1$ *defines an automorphism of* $\mathcal{U} \mathbin{\dot\otimes} \mathcal{U}$.

Notation. *If* φ *is an automorphism of* $\mathcal{A}$ *and* ψ *is an automorphism of* $\mathcal{B}$, *we write* $\varphi \mathbin{\dot\otimes} \psi$ *the automorphism of* $\mathcal{A} \mathbin{\dot\otimes} \mathcal{B}$ *defined for all* $A \in \mathcal{A}, B \in \mathcal{B}$ *as* $(\varphi \mathbin{\dot\otimes} \psi)(A \mathbin{\dot\otimes} B) := \varphi(A) \mathbin{\dot\otimes} \psi(B)$.

Definition 28 ***(normative pair)***
A pair $(\mathcal{A}, \mathcal{B})$ *of* $*$*-algebras is a* normative pair *whenever any automorphism* φ *of* $\mathcal{A}$ *can be extended into an automorphism* $\overline{\varphi}$ *of the* $*$*-algebra* $\mathcal{E}$ *generated by* $\mathcal{A} \cup \mathcal{B}$ *such that* $\overline{\varphi}(F) = F$ *for any* $F \in \mathcal{B} \subseteq \mathcal{E}$.

The two following propositions set the basis for a notion of acceptance/rejection independent of the representation of a word.

Proposition 29 ***(automorphic representations)***
Any two representations $W(t_0, \dots, t_n), W(u_0, \dots, u_n)$ *of a word* W *over* Σ *are automorphic: there exists an automorphism* φ *of* $(\mathtt{T}_\mathtt{c}^*)^{\otimes 1}$ *such that*

$$(Id_{\mathcal{U}} \mathbin{\dot\otimes} \varphi)\big(W(t_0, \dots, t_n)\big) = W(u_0, \dots, u_n)$$

Proof. Consider a bijection $f : \mathtt{T}_\mathtt{c} \to \mathtt{T}_\mathtt{c}$ such that $f(t_i) = u_i$ for all i. Then set $\varphi(v \bullet x \leftharpoonup w \bullet x) := f(v) \bullet x \leftharpoonup f(w) \bullet x$, extended by linearity. □

Proposition 30 ***(nilpotency and normative pairs)***
Let $(\mathcal{A}, \mathcal{B})$ *be a normative pair and* φ *an automorphism of* $\mathcal{A}$. *Let* $F \in \mathcal{U} \mathbin{\dot\otimes} \mathcal{A}$, $G \in \mathcal{U} \mathbin{\dot\otimes} \mathcal{B}$ *and let* $\psi := Id_{\mathcal{U}} \mathbin{\dot\otimes} \varphi$. *Then* GF *is nilpotent if and only if* $G\,\psi(F)$ *is nilpotent.*

Proof. Let $\overline{\varphi}$ be the extension of φ as in Definition 28 and $\overline{\psi} := \mathrm{Id}_{\mathcal{U}} \mathbin{\dot\otimes} \overline{\varphi}$.
We have for all $n \neq 0$ that $(G\psi(F))^n = (\overline{\psi}(G)\overline{\psi}(F))^n = (\overline{\psi}(GF))^n = \overline{\psi}((GF)^n)$.
By injectivity of $\overline{\psi}$, $(G\psi(F))^n = 0$ if and only if $(GF)^n = 0$. □

Corollary 31 ***(independence)***
If $\big((\mathtt{T}_\mathtt{c}^*)^{\otimes 1}, \mathcal{B}\big)$ *is a normative pair,* W *a word over* Σ *and* $F \in \mathcal{U} \mathbin{\dot\otimes} \mathcal{B}$. *The product of* F *with the representation of the word,* $FW(t_0, \dots, t_n)$, *is nilpotent for one choice of* $(t_0, \dots, t_n)$ *if and only if it is nilpotent for all choices of* $(t_0, \dots, t_n)$.

The basic components of the word and observation algebras we introduced earlier can be shown to form a normative pair.

Theorem 32
The pair $\big((\mathtt{T}_\mathtt{c}^*)^{\otimes 1}, \mathcal{S}\big)$ *is normative.*

Proof (sketch). By simple computations, the set

$$\mathcal{A} := \mathtt{Vect}\left\{ \sigma F \;\middle|\; \sigma \in \mathcal{S} \text{ and } F \in (\mathtt{T}_{\mathtt{c}}^{*})^{\otimes\infty} \right\}$$

can be shown to be a $*$-algebra $\mathcal{E}$, the $*$-algebra generated by $\mathcal{S} \cup (\mathtt{T}_{\mathtt{c}}^{*})^{\otimes 1}$.
As φ is an automorphism of $(\mathtt{T}_{\mathtt{c}}^{*})^{\otimes 1}$, it can be written as $\varphi(G \mathbin{\dot\otimes} I) = \psi(G) \mathbin{\dot\otimes} I$ for all G, with ψ an automorphism of $\mathtt{T}_{\mathtt{c}}^{*}$.
We set for $F = F_1 \mathbin{\dot\otimes} \cdots \mathbin{\dot\otimes} F_n \mathbin{\dot\otimes} I \in (\mathtt{T}_{\mathtt{c}}^{*})^{\otimes n}$, $\tilde\varphi(F) := \psi(F_1) \mathbin{\dot\otimes} \cdots \mathbin{\dot\otimes} \psi(F_n) \mathbin{\dot\otimes} I$ which extends into an automorphism of $(\mathtt{T}_{\mathtt{c}}^{*})^{\otimes\infty}$ by linearity. Finally, we extend $\tilde\varphi$ to $\mathcal{A}$ by $\overline{\varphi}(\sigma F) := \sigma\, \tilde\varphi(F)$. It is then easy to check that $\overline{\varphi}$ has the required properties. □

Remark. *Here we sketched a direct proof for brevity, but this can also be shown by involving a little more mathematical structure (actions of permutations on the unbounded tensor and crossed products) which would give a more synthetic proof.*

We can then define the notion of the language recognized by an observation, *via* Corollary 31.

Definition 33 ***(language of an observation)***
Let $\phi \in \mathcal{O}_{\Sigma}^{+}$ be an observation over Σ. The language recognized by ϕ *is the following set of words over Σ:*

$$\mathcal{L}(\phi) := \{\, W \text{ word over } \Sigma \mid \phi\, W(t_0, \ldots, t_n) \text{ nilpotent for any } (t_0, \ldots, t_n) \,\}$$

4 Wirings and Logarithmic Space

Now that we have defined our framework and showed how observations can compute, we study the complexity of deciding whenever an observation accepts a word (4.1), and how wirings can decide any language in (N)Logspace (4.2).

4.1 Soundness of Observations

The aim of this subsection is to prove the following theorem:

Theorem 34 ***(space soundness)***
Let $\phi \in \mathcal{O}_{\Sigma}^{+}$ be an observation over Σ.

- *$\mathcal{L}(\phi)$ is decidable in non-deterministic logarithmic space.*
- *If ϕ is isometric, then $\mathcal{L}(\phi)$ is decidable in deterministic logarithmic space.*

Actually, the result stands for the complements of these languages, but as co-NLogspace = NLogspace by the Immerman-Szelepcsényi theorem, this makes no difference.

The main tool for this purpose is the notion of *computation space*: finite dimensional subspaces of $\mathbb{V}_{\mathtt{c}}$ (Definition 16) on which we will be able to observe the behavior of certain wirings. It can be understood as the place where all the relevant computation takes place.

Definition 35 ***(separating space)***
A subspace E *of* $\mathbb{V}_{\mathtt{c}}$ *is* separating *for a wiring* F *whenever* $F(E) \subseteq E$ *and* $F^n(E) = 0$ *implies* $F^n = 0$ *.*

Observations are *finite* sums of wirings. We can naturally associate a finite-dimensional vector space to an observation and a finite set of closed terms.

Definition 36 ***(computation space)***
Let $\{ t_0, \dots, t_n \}$ *be a set of distinct closed terms and* $\phi \in \mathcal{O}_\Sigma^+$ *an observation. Let* $N(\phi)$ *be the smallest integer and* $\mathtt{S}(\phi)$ *the smallest (finite) set of closed terms such that* $\phi \in (\mathtt{S}(\phi)^* \mathbin{\dot\otimes} \Sigma^* \mathbin{\dot\otimes} \mathtt{LR}^*) \mathbin{\dot\otimes} \mathcal{S}_{N(\phi)}$ *.*
The computation space $\mathtt{Comp}_\phi(t_0, \dots, t_n)$ *is the subspace of* $\mathbb{V}_{\mathtt{c}}$ *generated by the terms*

$$s \bullet \mathtt{c} \bullet \mathtt{d} \bullet (a_1 \bullet \cdots \bullet a_{N(\phi)} \bullet \star)$$

where $s \in \mathtt{S}(\phi)$ *,* $\mathtt{c} \in \Sigma$ *,* $\mathtt{d} \in \mathtt{LR}$ *and the* $a_i \in \{ t_0, \dots, t_n \}$ *.*
The dimension of $\mathtt{Comp}_\phi(t_0, \dots, t_n)$ *is* $|\Sigma| 2 (n+1)^{N(\phi)} |\mathtt{S}(\phi)|$ *(where* $|A|$ *is the cardinal of* A *), which is polynomial in* n *.*

Lemma 37 ***(separation)***
For any observation ϕ *and any word* W *, the space* $\mathtt{Comp}_\phi(t_0, \dots, t_n)$ *is separating for the wiring* $\phi W(t_0, \dots, t_n)$ *.*

Proof (of Theorem 34). With these lemmas at hand, we can define the non-deterministic algorithm below. It takes as an input the representation $W(t_0, \dots, t_n)$ of a word W of length n .
ϕ being a constant, one can compute once and for all $N(\phi)$ and $\mathtt{S}(\phi)$.

1: $D \leftarrow 2|\mathtt{S}(\phi)|\,|\Sigma|(n+1)^{N(\phi)}$
2: $C \leftarrow 0$
3: pick a term $v \in \mathtt{Comp}_\phi(t_0, \dots, t_n)$
4: **while** $C \leq D$ **do**
5: **if** $(\phi W(t_0, \dots, t_n))(v) = 0$ **then**
6: **return** ACCEPT
7: **end if**
8: pick a term v' in $(\phi W(t_0, \dots, t_n))(v)$
9: $v \leftarrow v'$
10: $C \leftarrow C + 1$
11: **end while**
12: **return** REJECT

All computation paths (the "pick" at lines 3 and 8 being non-deterministic choices) accept if and only if $(\phi W(t_0, \dots, t_n))^n(\mathtt{Comp}_\phi(t_0, \dots, t_n)) = 0$ for some n lesser or equal to the dimension D of the computation space $\mathtt{Comp}_\phi(t_0, \dots, t_n)$. By Lemma 37, this is equivalent to $\phi W(t_0, \dots, t_n)$ being nilpotent.
The term chosen at lines 3 is representable by an integer of size at most D and is erased by the one chosen at line 8 every time we go through the **while**-loop. C and D are integers proportional to the dimension of the computation space, which is representable in logarithmic space in the size of the input (Definition 36).
The computation of $(\phi W(t_0, \dots, t_n))(v)$ at line 5 and 8 and can be performed in logarithmic space by Proposition 8, as we are unifying closed terms with linear terms.

Moreover, if ϕ is an isometric wiring, $(\phi W(t_0,\dots,t_n))(v)$ consists of a single term instead of a sum by Lemma 17, and there is therefore no non-deterministic choice to be made at line 8. It is then enough to run the algorithm enumerating all possible terms of $\mathtt{Comp}_\phi(t_0,\dots,t_n)$ at line 3 to determine the nilpotency of $\phi W(t_0,\dots,t_n)$. □

4.2 Completeness: Representing Pointer Machines as Wirings

To prove the converse of Theorem 34, we prove that wirings can encode a special kind of read-only multi-head Turing Machine: pointers machines. The definition of this model will be guided by our understanding of the computation of wirings: they won't have the ability to write and acceptance will be defined as termination of all paths of computation. For a survey of this topic, one may consult the first author's thesis [21, Chap.4], the main novelty of this part of our work is to notice that reversible computation is represented by isometric operators.

Definition 38 ***(pointer machine)***
A pointer machine *over an alphabet* Σ *is a tuple* $(N,\mathtt{S},\Delta)$ *where*

- $N\neq 0$ *is an integer, the* number of pointers*,*
- $\mathtt{S}$ *is a finite set, the* states *of the machine,*
- $\Delta\subseteq(\mathtt{S}\times\Sigma\times\mathtt{LR})\times(\mathtt{S}\times\Sigma\times\mathtt{LR})\times\mathfrak{S}_N$ *, the* transitions *of the machine (we will write* $(s,\mathtt{c},\mathtt{d})\to(s',\mathtt{c}',\mathtt{d}')\times\sigma$ *the transitions, for readability).*

A pointer machine will be called deterministic *if for any* $A\in\mathtt{S}\times\Sigma\times\mathtt{LR}$*, there is at most one* $B\in\mathtt{S}\times\Sigma\times\mathtt{LR}$ *and one* $\sigma\in\mathfrak{S}_N$ *such that* $A\to B\times\sigma\in\Delta$*. In that case we can see* Δ *as a partial function, and we say that* M *is* reversible *if* Δ *is a partial injection.*

We call the first of the N pointers the *main* pointer, it is the only one that can move. The other pointers are referred to as the *auxiliary* pointers. An auxiliary pointer will be able to become the main pointer during the computation thanks to permutations.

Definition 39 ***(configuration)***
Given the length n *of a word* $W=\star\mathtt{c}_1\dots\mathtt{c}_n$ *over* Σ *and a pointer machine* $M=(N,\mathtt{S},\Delta)$*, a* configuration *of* (M,n) *is an element of*

$$\mathtt{S}\times\Sigma\times\mathtt{LR}\times\{0,1,\dots,n\}^N$$

The element of $\mathtt{S}$ is the state of the machine and the element of Σ is the letter the main pointer points at. The element of $\mathtt{LR}$ is the direction of the next move of the main pointer, and the elements of $\{0,1,\dots,n\}^N$ correspond to the positions of the (main and auxiliary) pointers on the input.

As the input tape is considered cyclic with a special symbol marking the beginning of the word (recall Definition 24), the pointer positions are integers *modulo* $n+1$ for an input word of length n.

Definition 40 ***(transition)***
Let W *be a word and* $M = (N, \mathtt{S}, \Delta)$ *be a pointer machine. A* transition *of* M *on input* W *is a triple of configurations*

$$s, \mathtt{c}, \mathtt{d}, (p_1, \dots, p_N) \xrightarrow{\texttt{MOVE}} s, \mathtt{c}', \overline{\mathtt{d}}, (p'_1, \dots, p'_N) \xrightarrow{\texttt{SWAP}} s', \mathtt{c}'', \mathtt{d}', (p'_{\sigma(1)}, \dots, p'_{\sigma(N)})$$

such that

1. *if* $\mathtt{d} \in \mathtt{LR}$, $\overline{\mathtt{d}}$ *is the other element of* $\mathtt{LR}$,
2. $p'_1 = p_1 + 1$ *if* $\mathtt{d} = \mathtt{R}$ *and* $p'_1 = p_1 - 1$ *if* $\mathtt{d} = \mathtt{L}$,
3. $p'_i = p_i$ *for* $i \neq 1$,
4. $\mathtt{c}$ *is the letter at position* p_1 *and* $\mathtt{c}'$ *is the letter at position* p'_1,
5. *and* $(s, \mathtt{c}', \overline{\mathtt{d}}) \to (s', \mathtt{c}'', \mathtt{d}') \times \sigma$ *belongs to* Δ.

There is no constraint on c'', but every time this value differs from the letter pointed by $p'_{\sigma(1)}$, the computation will halt on the next `MOVE` phase, because there is a mismatch between the value that is supposed to have been read and the actual bit of W stored at this position, and that would contradict the first part of item 4. In terms of wirings, the `MOVE` phase corresponds to the application of the representation of the word, whereas the `SWAP` phase corresponds to the application of the observation.

Definition 41 ***(acceptance)***
We say that M *accepts* W *if any sequence of transitions* $\left(C_i \xrightarrow{\texttt{MOVE}} C'_i \xrightarrow{\texttt{SWAP}} C''_i\right)$ *such that* $C''_i = C_{i+1}$ *for all* i *is necessarily finite.*
We write $\mathcal{L}(M)$ *the set of words accepted by* M.

This means informally: we consider that a pointer machine accepts a word if it cannot ever loop, from whatever configuration it starts from. That a lot of paths of computation accepts "wrongly" is no worry, since only rejection is meaningful: our pointer machines compute in a "universally non-deterministic" way, to stick to the acceptance condition of wirings, nilpotency.

Proposition 42 ***(space and pointer machines)***
If $L \in$ NLogspace, *then there exist a pointer machine* M *such that* $\mathcal{L}(M) = L$. *Moreover, if* $L \in$ Logspace *then* M *can be chosen to be reversible.*

Proof (sketch). It is well-known that read-only Turing Machines – or equivalently (non-)Deterministic Multi-Head Finite Automata – characterize (N)Logspace [22]. It takes little effort to see that our pointer machines are just a reasonable rearrangement of this model, since it is always possible to encode the missing information in the states of the machine.
That acceptance and rejections are "reversed" is harmless in the deterministic (or equivalently reversible [23]) case, and uses that co-NLogspace = NLogspace to get the expected result in the non-deterministic case. □

As we said, our pointer machines are designed to be easily simulated by wirings, so that we get the expected result almost for free.

Theorem 43 ***(space completeness)***
If $L \in \mathrm{NLogspace}$*, then there exist an observation* $\phi \in \mathcal{O}_\Sigma^+$ *such that* $\mathcal{L}(\phi) = L$*. Moreover, if* $L \in \mathrm{Logspace}$ *then* ϕ *is an isometric wiring.*

Proof. By Proposition 42, there exists a pointer machine $M = (N, \mathtt{S}, \Delta)$ such that $\mathcal{L}(M) = L$. We associate to the set $\mathtt{S}$ a set of distinct closed terms $[\mathtt{S}]$ and write $[s]$ the term associated to s. To any element $D = (s, \mathtt{c}, \mathtt{d}) \to (s', \mathtt{c}', \mathtt{d}') \times \sigma$ of Δ we associate the flow

$$[D] := ([s']_\bullet \mathtt{c}'_\bullet \mathtt{d}' \leftharpoondown [s]_\bullet \mathtt{c}_\bullet \mathtt{d}) \mathbin{\dot\otimes} [\sigma] \ \in ([\mathtt{S}]^* \mathbin{\dot\otimes} \Sigma^* \mathbin{\dot\otimes} \mathtt{LR}^*) \mathbin{\dot\otimes} \mathcal{S}_n \ \subseteq \ \mathcal{O}_\Sigma^+$$

and we define the observation $[M] \in \mathcal{O}_\Sigma^+$ as $\sum_{D \in \Delta} [D]$.
One can easily check that this translation preserves the language recognized (there is even a step by step simulation of the computation on the word W by the wiring $[M]W(t_0, \dots, t_n)$) and relates reversibility with isometricity: in fact, M is reversible if and only if $[M]$ is an isometric wiring. Then, if $L \in \mathrm{Logspace}$, M is deterministic and can always be chosen to be reversible [23]. □

Discussion

The language of the unification algebra gives us a twofold point of view on computation, either through algebraic structures (that are described finitely by wirings) or pointer machines. We may therefore start exploring possible variations of the construction, combining intuitions from both worlds.

For instance, the choice of a normative pair can affect the expressivity of the construction: the more restrictive the notion of representation of a word is, the more liberal that of an observation can become, as suggested by T. Seiller. Whether and how this can affect the corresponding complexity class is definitely a direction for future work.

Another pending question about this approach to complexity classes is to delimit the minimal prerequisites of the construction, its core.
Earlier works [13,14,15] made use of von Neumann algebras to get a setting that is expressive enough, we ligthen the construction by using simpler objects. Yet, the possibility of representing the action of permutations on a unbounded tensor product is a common denominator that seems deeply related to logarithmic space and pointer machines.

The logical counterpart of this work also needs clarifying. Indeed, the idea of representation of words comes directly from proof-theory, while the notion of observation does not seem to correspond to any known logical construction.

Finally, execution in our setting being based on iteration of matching, which is computable efficiently by a parallel machine, it seems possible to relate our modelisation with parallel computation.

References

1. Girard, J.Y.: Linear logic. Theoretical Computer Science **50** (1987) 1–102
2. Girard, J.Y., Scedrov, A., Scott, P.J.: Bounded Linear Logic: A Modular Approach to Polynomial Time Computability. Theoretical Computer Science **97**(1) (1992) 1–66
3. Girard, J.Y.: Light linear logic. In Leivant, D., ed.: Logic and Computational Complexity. Volume 960 of Lecture Notes in Computer Science. (1995) 145–176
4. Schöpp, U.: Stratified Bounded Affine Logic for Logarithmic Space. In: LICS, IEEE Computer Society (2007) 411–420
5. Dal Lago, U., Hofmann, M.: Bounded Linear Logic, Revisited. Logical Methods in Computer Science **6**(4) (2010)
6. Gaboardi, M., Marion, J.Y., Ronchi Della Rocca, S.: An Implicit Characterization of PSPACE. ACM Transactions on Computational Logic **13**(2) (2012) 18
7. Baillot, P., Mazza, D.: Linear logic by levels and bounded time complexity. Theoretical Computer Science **411**(2) (2010) 470–503
8. Girard, J.Y.: Towards a Geometry of Interaction. In: Proceedings of the AMS Conference on Categories, Logic and Computer Science. (1989)
9. Asperti, A., Danos, V., Laneve, C., Regnier, L.: Paths in the lambda-calculus. In: LICS, IEEE Computer Society (1994) 426–436
10. Laurent, O.: A token machine for full geometry of interaction (extended abstract). In Abramsky, S., ed.: Typed Lambda Calculi and Applications. Volume 2044 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2001) 283–297
11. Girard, J.Y.: Geometry of interaction 1: Interpretation of System F. Studies in Logic and the Foundations of Mathematics **127** (1989) 221–260
12. Baillot, P., Pedicini, M.: Elementary Complexity and Geometry of Interaction. Fundamenta Informaticae **45**(1-2) (2001) 1–31
13. Girard, J.Y.: Normativity in Logic. In: Epistemology versus Ontology. Volume 27 of Logic, Epistemology, and the Unity of Science. Springer (2012) 243–263
14. Aubert, C., Seiller, T.: Characterizing co-NL by a group action. Arxiv preprint **abs/1209.3422** (2012)
15. Aubert, C., Seiller, T.: Logarithmic Space and Permutations. Arxiv preprint **abs/1301.3189** (2013)
16. Girard, J.Y.: Geometry of Interaction III: Accommodating the Additives. In: Advances in Linear Logic, LNS 222,CUP, 329–389. (1995) 329–389
17. Girard, J.Y.: Three lightings of logic (Invited Talk). In Ronchi Della Rocca, S., ed.: CSL. Volume 23 of Leibniz International Proceedings in Informatics., Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013) 11–23
18. Knight, K.: Unification: A Multidisciplinary Survey. ACM Computing Surveys **21**(1) (1989) 93–124
19. Dwork, C., Kanellakis, P.C., Mitchell, J.C.: On the sequential nature of unification. Journal of Logic Programming **1**(1) (1984) 35–50
20. Dwork, C., Kanellakis, P.C., Stockmeyer, L.J.: Parallel Algorithms for Term Matching. SIAM Journal on Computing **17**(4) (1988) 711–731
21. Aubert, C.: Linear Logic and Sub-polynomial Classes of Complexity. PhD thesis, Université Paris 13 – Sorbonne Paris Cité (November 2013)
22. Hartmanis, J.: On Non-Determinancy in Simple Computing Devices. Acta Informatica **1**(4) (1972) 336–344
23. Lange, K.J., McKenzie, P., Tapp, A.: Reversible Space Equals Deterministic Space. Journal of Computer and System Sciences **60**(2) (2000) 354–367