Unification and logarithmic space

Clément Aubert & Marc Bagnol

Institut de Mathématiques de Marseille

Abstract. We present an algebraic characterization of the complexity classes L and NL of deterministic and non-deterministic logarithmic space, using an algebra with a composition law based on unification. We show that computation can be modeled in it by means of specific subalgebras related to groups of finite permutations.
We also show that the construction can be related with a notion pointer machines, a model of logarithmic space computation.

Introduction

Proof theory and complexity theory. Since the introduction of so-called light logics [12], several author contributed to the topic of implicit complexity theory using tools coming from proof theory and more specifically linear logic [9].

The study of geometrical properties of proofnets [14], in terms for instance of stratitification of exponential boxes [3] led to a clearer understanding of the time complexity of the cut-elimination procedure. Space complexity has also been studied from this perspective [26,8].

Geometry of Interaction. As the study of cut-elimination has grown as a central topic in proof theory, its mathematical modelling became of great interest. The Geometry of Interaction [10] research program led to models of cut elimination in terms of paths in proofnets [1], token machines [20] and operator algebras [10,11,15]. The relationship of these ideas with complexity theory have also started to be investigated [25,4].

The tools shaped by this approach have also been used to study directly complexity theory from an algebraic point of view [16,2,27]. These works are the main source of inspiration for this article.

Unification. The unification technique has been used in [13,4] to provide a setting where one can model the cut-elimination procedure in a finitary way: first-order terms with variables offer a way to manipulate infinite sets (of instances of terms) with a finite language.

It turns out that this language is expressive enough to encode various algebraic structures, including a notion of unbounded tensor product and a representation of finite permutation groups.

Pointer machines. The study of space efficient computations lead to consider pointer machines as a mathematical model of computing [18,19].

In [16], how operators simulate computation is briefly explained in terms of a pointer machines, an idea that has been made precise by the later [2,27].

Organisation of this article. In section 1 we review some classical results on unification of first-order terms and use them to build an algebra that will constitute our computationnal setting.

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

Finally, we show in section 4 that our construction captures exactly logarithmic space computation, and we relate it with a notion of pointer machine.

1 The unification algebra

1.1 Unification

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

This topic was introduced by Herbrand in his thesis [17], but became really widespread after the work of J. A. Robinson on automated theorem proving [24]. 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 (first-order) terms,
can they be "made equal" by replacing their variables?

To make more things concrete, we set a specific set of terms for the rest of this article.

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

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

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

For any $t \in \mathbb{T}$*, we will write* $\mathtt{Var}(t)$ *the set of variables occuring in* t*. We say that a term is closed whenever* $\mathtt{Var}(t) = \varnothing$*, and denote the set of closed terms as* $\mathbb{T}^{\mathsf{c}}$*.*

Notation. *We will write the binary function symbol as* right associating *to avoid an excess of parenthesis:*

$$t \bullet u \bullet v \ := \ t \bullet (u \bullet v)$$

Definition 2 ***(substitution).***
A substitution if a map $\theta : \mathbb{V} \to \mathbb{T}$ *such that the set* $\mathtt{Dom}(\theta) := \{ v \in \mathbb{V} \mid \theta(v) \neq v \}$ *(the* domain *of* θ*) is finite. A substitution with domain* $\{ x_1, \dots, x_n \}$ *such that*

$\theta(x_1) = u_1, \dots, \theta(x_n) = u_n$ can therefore be written as $\{ x_1 \mapsto u_1 ; \dots ; x_n \mapsto u_n \}$.

If $t \in \mathbb{T}$ is a term we write $t.\theta$ the term t where any occurence 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*

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

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

Definition 3 ***(vocabulary).***
A renaming *is a substitution α such that $\alpha(\mathbb{V}) \subseteq \mathbb{V}$ and that is bijective.*

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

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

A substitution θ is idempotent *if $\theta;\theta = \theta$.*

Theorem 4 ***.***
The following properties hold for any substitutions θ, ψ:

- *The only invertible substitutions are renamings.*
- *Every substitution is equal up to renaming to an idempotent substitution.*
- *If θ is an instance of ψ and ψ is an instance of θ, then they are equal up to renaming.*

To allow easier manipulation, the problem of unifying *two* terms needs to be generalized into the problem of simultaneously unifying several pairs of terms.

Definition 5 ***(unification).***
A unification problem *is finite set of pair of terms $P = \{ t_i \overset{?}{=} u_i \}$.*
It is said to be unifiable if it has a unifier*: a substitution θ such that*

$$u_i.\theta = t_i.\theta \quad \textit{for all } i$$

θ is a most general unifier (MGU) of *P if any other unifier of P is an instance of θ.*

In particular, we say that two terms t, u are unifiable *if there exists a substitution θ such that*

$$t.\theta = u.\theta$$

Remark. *It follows from proposition 4 that any two MGU of the same unification problem are equal up to renaming.*

We will be interested mostly in the weaker variant of unification where one can first rename terms so that they 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 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. However 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-unifyability.

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

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

We can determine if a unification problem has a solution. We note the following fact, that will be useful in the next section: solving a unification problem can be done incrementally.

Notation. *If $P = \{\, t_i \overset{?}{=} u_i \,\}$ is a unification problem and θ a substitution, we write*

$$P.\theta := \{\, t_i.\theta \overset{?}{=} u_i.\theta \,\}$$

Lemma 8 ***(partial unification).***
Let $P = Q \uplus R$ ($\uplus$ denotes disjoint union) be a unification problem. The following statements are equivalent:

- *P is unifiable*
- *Q is unifiable with MGU θ and $R.\theta$ is unifiable with MGU ψ*

In that case, we have moreover that $\theta;\psi$ is a MGU of P.

J. A. Robinson was the first to prove the existence of MGU for two unifiable terms and derived from this a deterministic unification procedure.His procedure is however quite inefficient, with potential exponential blowups in some cases.

It turns out that the general unification problem can be solved in linear time, which was discovered independently in [23] and [22], improving on the almost-linear algorithm thus far designed.

Unification was first thought to be a NLogspace-complete problem. However, an error was eventually found in the note [21] claiming this fact, which eventually led to a determination of the intrinsic complexity of the problem: unification is indeed Ptime-complete under logarithmic space reductions [6].

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.

Theorem 9 ***(logarithmic space [7, lemma 20]).***
Wether two linear terms t, u with disjoint sets of variables are unifiable, and if so finding a MGU, can be computed in logarithmic space on a deterministic Turing machine.

Moreover, their MGU is of size at most the sum of the sizes of t and u.

Actually, the lemma in the article states that the problem is in NC^1, a complexity class of parallel computations known to be included in Logspace.

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, that is the basis of the construction.

Definition 10 ***(flows).***
A flow is an oriented pair written $t \leftharpoonup u$ of terms such that $\mathtt{Var}(t) = \mathtt{Var}(u)$.
Flows are considered up to renaming: for any variable renaming α we have $t \leftharpoonup u = t.\alpha \leftharpoonup u.\alpha$.
We will write $\mathcal{F}$ the set of (equivalence classes of) flows.
We set $(t \leftharpoonup u)^\dagger := u \leftharpoonup t$ so that $(.)^\dagger$ is an involution $\mathcal{F}$.
Finally, define $1 := x \leftharpoonup x$.

A flow $u \leftharpoonup v$ can be thought of as a `‘match ... with v -> u’` in a ML-style language. The composition of flows follows intuition.

Definition 11 ***(product of flows).***
Let $u \leftharpoonup v \in \mathcal{F}$ and $t \leftharpoonup w \in \mathcal{F}$. Suppose we have chosen two representatives of the renaming classes such that their sets of variables are disjoint.
The product of $u \leftharpoonup v$ and $t \leftharpoonup w$ is defined if v, t are unifiable with MGU θ and in that case

$$l_1 \, l_2 := u.\theta \leftharpoonup w.\theta$$

Definition 12 ***(action on closed terms).***
If $t \in \mathbb{T}^{\mathsf{c}}$ is a closed term, $(u \leftharpoonup v)(t)$ is defined whenever t and v are unifiable, with MGU θ, in that case $(u \leftharpoonup v)(t) := u.\theta$

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 the 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 sides are defined at the same time.

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

Definition 13 ***(wirings).***
We call wirings *$\mathbb{C}$-linear combinations (formally: the set of almost-everywhere null functions from $\mathcal{F}$ to $\mathbb{C}$) of elements of $\mathcal{F}$, endowed with:*

$$\Big(\sum_i \lambda_i\, l_i\Big)\Big(\sum_j \mu_j\, k_j\Big) := \sum_{\substack{i,j \text{ such that} \\ (l_i k_j) \text{ is defined}}} \lambda_i \mu_j (l_i\, k_j)$$

$$\text{and} \qquad \Big(\sum_i \lambda_i\, l_i\Big)^{\dagger} := \sum_i \overline{\lambda}_i\, l_i^{\dagger}$$

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

Remark. *$\mathcal{U}$ is indeed a unital $*$-algebra: the only delicate point would be the associativity of the product, which is a consequence of lemma 8.*

Definition 14 ***(partial isometries and orthogonal projections).***
A partial isometry *is a wiring satisfying $UU^{\dagger}U = U$.*
An orthogonal projection *is a wiring satisfying $U^2 = U$ and $U^{\dagger} = U$.*

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

Definition 15 ***(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 will write E^{+} the set of concrete wirings of E.

Isometric wirings enjoy a direct characterisation.

Theorem 16 ***(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 the t_i pairwise disjoint.

It will be useful to consider the action of wirings on closed terms. for this purpose we extend definition 12 to wirings.

Definition 17 ***(action on closed terms).***
Let $\mathbb{H}^{\mathsf{c}}$ be the free vector space over $\mathbb{T}^{\mathsf{c}}$.
Wirings act on base vectors of $\mathbb{H}^{\mathsf{c}}$ the following way

$$\Big(\sum_i \lambda_i\, l_i\Big)(t) := \sum_{\substack{i \text{ such that} \\ l_i(t) \text{ is defined}}} \lambda_i \big(l_i(t)\big) \quad \in \mathbb{H}^{\mathsf{c}}$$

which extends by linearity into an action on the whole $\mathbb{H}^{\mathsf{c}}$.

Note that isometric wirings have a particular behaviour in terms of this action.

Lemma 18 ***(isometric action).***
Let F be a 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 $\mathbb{H}^{\mathsf{c}}$).

1.3 Permutations and tensor product

Notations. *Given any set of wirings E, we write respectively* $\mathtt{Vect}(E)$, $\mathtt{Alg}(E)$ *and* $\mathtt{Alg}^\dagger(E)$ *the vector space, the algebra and the $*$-algebra generated by E. Moreover, we set*

$$\mathbf{1} := \mathbb{C}.1 = \mathtt{Vect}\{\, 1 \,\} = \mathtt{Alg}^\dagger\{\, 1 \,\} \qquad\qquad u \leftrightharpoons v := u \leftharpoonup v + v \leftharpoonup u$$

We will always write "$*$-algebras" instead of the more correct "$*$-subalgebras of $\mathcal{U}$", for brevity.

Two important feature of $\mathcal{U}$ for this work are the possibility of representing (unbounded) tensor products and finite-support permutations of $\mathbb{N}$.

Definition 19 ***(tensor product).***
Let $u \leftharpoonup v$ and $t \leftharpoonup w$ be two flows. Suppose we have chosen representatives of the renaming classes that have their sets of variables disjoint. We define their tensor product as

$$(u \leftharpoonup v) \otimes (t \leftharpoonup w) := u {\bullet} t \leftharpoonup v {\bullet} w$$

The operation is extended to wirings by bilinearity.

Given two $*$-algebras $\mathcal{A}, \mathcal{B}$, we define their tensor product as the $*$-algebra

$$\mathcal{A} \otimes \mathcal{B} := \mathtt{Vect}\{\, F \otimes G \mid F \in \mathcal{A},\ G \in \mathcal{B} \,\}$$

This actually defines an embedding of the algebraic tensor product of $*$-algebras into $\mathcal{U}$. Which means in particular that $(F \otimes G)(P \otimes Q) = (FP) \otimes (GQ)$.

Notation. *We carry on the convention for the binary function symbol and write the tensor product of $*$-algebras as* right associating *to avoid an excess of parenthesis:*

$$\mathcal{A} \otimes \mathcal{B} \otimes \mathcal{C} := \mathcal{A} \otimes (\mathcal{B} \otimes \mathcal{C})$$

Definition 20 ***(unbounded tensor).***
Let $\mathcal{A}$ be a $*$-algebra. We define the $*$-algebras $\mathcal{A}^n$ for all $n \in \mathbb{N}$ as

$$\mathcal{A}^0 := \mathbf{1} \quad \textit{and} \quad \mathcal{A}^{n+1} := \mathcal{A} \otimes \mathcal{A}^n$$

and the $*$-algebras $\mathcal{A}^\infty := \bigcup_{n \in N} \mathcal{A}^n$ *and* $\mathcal{A}^{\leq N} := \bigcup_{n \in \{\, 1, \ldots, N \,\}}^{N} \mathcal{A}^n$.

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

Notations. *If $\sigma \in \mathfrak{S}_n$, we define $\sigma_{+k} \in \mathfrak{S}_{n+k}$ as the permutation σ extended to $\{\, 1, \ldots, n, \ldots, n+k \,\}$ letting $\sigma_{+k}(n+i) := n+i$.*
We also write $1_k := Id_{\{1,\ldots,k\}} \in \mathfrak{S}_k$.

Definition 21 ***(representation).***
To a permutation $\sigma \in \mathfrak{S}_n$ *we associate the flow*

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

We will omit the brackets when there is no ambiguity.

Lemma 22 ***(representation).***
For $\sigma \in \mathfrak{S}_n$ *and* $\tau \in \mathfrak{S}_{n+k}$ *we have*

$$[\sigma_{+k}] = [\sigma] \qquad [1_{n+k}] = [1_{n+k}][\sigma] \qquad [\sigma_{+k} \circ \tau] = [\sigma][\tau]$$

$$\textit{and} \qquad [\sigma^{-1}] = [\sigma]^\dagger$$

Definition 23 ***(permutation algebra).***
For $n \in \mathbb{N}$ *we set* $[\mathfrak{S}_n] := \{\, [\sigma] \mid \sigma \in \mathfrak{S}_n \,\}$ *and* $\mathcal{S}_n := \mathtt{Alg}^\dagger[\mathfrak{S}_n] = \mathtt{Vect}[\mathfrak{S}_n]$.
We define then $\mathcal{S} := \bigcup_{n \in \mathbb{N}} \mathcal{S}_n$ *, which we call the* permutation algebra*.*

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 [11,13]. However, no knowledge of these topics is strictly needed and we can give a direct presentation of the notion.

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

Definition 24 ***(word algebra).***
To a finite set Σ *of constant symbols, we associate the* $*$*-algebra*

$$[\Sigma] := \mathtt{Vect}\{\, \mathtt{c} \leftharpoonup \mathtt{d} \mid \mathtt{c}, \mathtt{d} \in \Sigma \,\}$$

When it is clear from the context, we will simply write Σ *in place of* $[\Sigma]$.
The word algebra associated to Σ *is the* $*$*-algebra defined as*

$$\mathcal{W}_\Sigma := \mathbf{1} \otimes \Sigma \otimes \mathtt{LR} \otimes (\mathbb{T}^{\mathtt{c}})^1$$

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 $\star\mathtt{0010} = \mathtt{01}\star\mathtt{00} = \mathtt{0}\star\mathtt{001} = \cdots$
We consider therefore from now on that the alphabet we consider contains the symbol $\star$.

Definition 25 ***(word representation).***
Let $W = \star\mathtt{c}_1 \ldots \mathtt{c}_n$ *be a word over* Σ *of length* n .

A representation $W(t_0, t_1, \dots, t_n) \in \mathcal{W}_\Sigma^+$ *of* W *is an isometric wiring determined by the distinct closed terms* $t_0, t_1, \dots, t_n$ *, defined as*

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

Observations will stand for the machines accepting and refusing words. They lie in a particular $*$-algebra based on the representation of permutations presented in section 1.3.

Definition 26 ***(observation algebra).***
An observation *over a finite set of symbols* Σ *is any element of* $\mathcal{O}_\Sigma^+$ *where* $\mathcal{O}_\Sigma := \mathbb{T}^{\mathtt{c}} \otimes \Sigma \otimes \mathtt{LR} \otimes \mathcal{S}$ *, i.e. a* finite *sum of flows of the form*

$$(s' \bullet \mathtt{c}' \bullet \mathtt{d}' \leftharpoonup s \bullet \mathtt{c} \bullet \mathtt{d}) \otimes \sigma$$

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

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

3 Normativity

We are going to define the way observations accept and reject words. Indeed there is a issue with the notion of representation: 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 25, different choices of closed terms lead to different representations).
Therefore one has to ensure that acceptation or rejection is independent of the representation, so that the construction makes sense.

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

- $\varphi(FG) = \varphi(F)\varphi(G)$
- $\varphi(F^\dagger) = \varphi(F)^\dagger$
- φ *is injective*

If there is a partial isometry $U \in \mathcal{A}$ *such that* $\varphi(a) = u^\dagger a u$ *for all* a *, then we say that* φ *is an* inner *automorphism.*

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* $\mathtt{Alg}^\dagger(\mathcal{A} \cup \mathcal{B})$ *such that* $\overline{\varphi}(F) = F$ *for any* $F \in \mathcal{B} \subseteq \mathtt{Alg}^\dagger(\mathcal{A} \cup \mathcal{B})$ *.*

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

Theorem 29 ***(automorphic representations).***
Any two representations $W(t_i), W(u_i)$ of a word W over Σ are automorphic: there exists an automorphism φ of $(\mathbb{T}^{\mathsf{c}})^1$ such that

$$(Id_{\mathbf{1}\otimes\Sigma\otimes\mathtt{LR}}\otimes\varphi)\big(W(t_i)\big) = W(u_i)$$

Proof. Take as φ the inner automorphism associated with the isometric wiring

$$\sum_i t_i \bullet x \leftharpoonup u_i \bullet x \ \in \ (\mathbb{T}^{\mathsf{c}})^1$$

$\square$

Theorem 30 ***(nilpotency and normative pairs).***
Let $(\mathcal{A},\mathcal{B})$ be a normative pair, $F\in\mathcal{A}$, $G\in\mathcal{B}$ and φ an automorphism of $\mathcal{A}$, then GF is nilpotent if and only if $G\,\varphi(F)$ is nilpotent.

Proof. Let $\overline{\varphi}$ be the extension of φ as in defintion 28. We have for all $n\geq 1$ that $(G\varphi(F))^n = (\overline{\varphi}(G)\overline{\varphi}(F))^n = (\overline{\varphi}(GF))^n = \overline{\varphi}((GF)^n)$.
By injectivity of $\overline{\varphi}$, $(\varphi(G)F)^n = 0$ if and only if $(GF)^n = 0$. $\square$

Corollary 31 ***(independance).***
If $\big((\mathbb{T}^{\mathsf{c}})^1,\mathcal{B}\big)$ is a normative pair, W a word over Σ and $F\in\mathbb{T}^{\mathsf{c}}\otimes\Sigma\otimes\mathtt{LR}\otimes\mathcal{B}$, then $FW(t_i)$ is nilpotent for one choice of t_i if and only if it is nilpotent for all choices of t_i.

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

Theorem 32 ***(a normative pair).***
For any Σ, $\big((\mathbb{T}^{\mathsf{c}})^1,\mathcal{S}\big)$ is a normative pair

Proof (sketch). By simple computations, the set

$$\mathcal{A} := \mathtt{Vect}\,\{\,\sigma F \mid \sigma\in\mathcal{S} \text{ and } F\in(\mathbb{T}^{\mathsf{c}})^{\infty}\,\}$$

can be shown to be a $*$-algebra, so that $\mathtt{Alg}^{\dagger}(\mathcal{S}\cup(\mathbb{T}^{\mathsf{c}})^1) = \mathtt{Alg}^{\dagger}(\mathcal{S}\cup(\mathbb{T}^{\mathsf{c}})^{\infty}) = \mathcal{A}$.
Now if φ is an automorphism of $(\mathbb{T}^{\mathsf{c}})^1$, it can be written as $\varphi(t\bullet x) = \psi(t)\bullet x$ with ψ an automorphism of $\mathbb{T}^{\mathsf{c}}$. We define for $F = F_1\otimes\cdots\otimes F_n\otimes 1\in(\mathbb{T}^{\mathsf{c}})^n$, $\tilde{\phi}(F) := \psi(F_1)\otimes\cdots\otimes\psi(F_n)\otimes 1$ which gives an automorphism of $(\mathbb{T}^{\mathsf{c}})^{\infty}$ by linearity. Finally we extend $\tilde{\varphi}$ to $\mathcal{A}$ by $\overline{\varphi}(\sigma F) := \sigma\tilde{\varphi}(F)$ $\square$

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.*

Therefore the notion of the language recognized by an observation makes sense, *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*

$$\mathbf{L}(\phi) := \{\, W \text{ word over } \Sigma \mid \phi\, W(t_i) \text{ nilpotent for any choice of } (t_i) \,\}$$

4 Logarithmic space

4.1 Soundness

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

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

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

The main tool for this purpose is the notion of *computation space*: a finite dimensional subspace of $\mathbb{H}^{\mathsf{c}}$ on which we will be able to observe the behaviour of our wirings.

Definition 35 ***(separating space).***
A subspace E of $\mathbb{H}^{\mathsf{c}}$ is separating *for a wiring F whenever it is stable by F and $F^n(E) = 0$ implies $F^n = 0$.*

As observations are *finite* sums of wirings, we are able to 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 $\phi \in \Sigma \otimes \mathtt{LR} \otimes \mathcal{S}_{N(\phi)} \otimes \mathtt{S}(\phi)$.
The computation space $\mathtt{Comp}_{\phi}(t_0 \dots t_n)$ *is the subspace of $\mathbb{H}^{\mathsf{c}}$ generated by the terms*

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

where $s \in \mathtt{S}(\phi)$, $\mathtt{c} \in \Sigma$, $d \in \mathtt{LR}$ and the $a_i \in \{\, t_0 \dots t_n \,\}$.

Lemma 37 ***(dimension).***
The dimension of $\mathtt{Comp}_{\phi}(t_0 \dots t_n)$ is $2|\mathtt{S}(\phi)|\,|\Sigma|(n+1)^{N(\phi)}$.

Lemma 38 ***(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 following non-deterministic algorithm

```
1: compute $N(\phi)$ and $S(\phi)$
2: $D \leftarrow 2|\mathtt{S}(\phi)|\,|\Sigma|(n+1)^{N(\phi)}$
3: $C \leftarrow 0$
4: pick a term $t \in \mathtt{Comp}_{\phi}(t_i)$
5: while $C \leq D$ do
6:   if $(\phi W(t_i))(v)$ then
7:     return ACCEPT
8:   end if
9:   pick a vector $v'$ in $(\phi W(t_i))(v)$
10:  $v \leftarrow v'$
11: end while
12: return REJECT
```

All computation paths accept if and only if $(\phi W(t_i))^n(\mathtt{Comp}_{\phi}(t_i)) = 0$ for some n lesser or equal to the dimension of the computation space. By lemma 38, this is equivalent to $\phi W(t_i)$ nilpotent.
The algorithm can run in logarithmic space, because it only needs to store a vector of a vector space of dimension $D = 2|\mathtt{S}(\phi)|\,|\Sigma|(n+1)^{N(\phi)}$ (which can be represented as an integer lesser or equal to D), and the two integers C, D. The computation of $(\phi W(t_i))(v)$ at line 6 and 9 and can be performed in logarithmic space by theorem 9.

Moreover, if ϕ is an isometric wiring, $(\phi W(t_i))(v)$ consists of a single vector instead of a sum, and there is therefore no non-deterministic choice to be made at line 9. It is enough to run the algorithm enumarating all possible vectors of $\mathtt{Comp}_{\phi}(t_i)$ at line 4 to determine the nilpotency of $\phi W(t_i)$. □

4.2 Pointer machines and completeness

To prove the converse of theorem 34, we introduce a notion of pointer machine that can easily be encoded as wirings. Apart from the result, this also provides a way of seeing how wiring are computing.
Pointer machines [5] are theoretical devices that aim at modelling computation in a way that varies from turing machines: the data is a read-only sequence of symbols, and there is no memory tape. Instead, the machine manipulates pointers to various point of the input.

Definition 39 ***(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.*

Definition 40 ***(configuration).***
Given an input word w of length n and an pointer machine $M = (N, \mathtt{S}, \Delta)$, a configuration *of (M, w) is an element of*

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

Let us explain the intuitive meaning of a configuration. We call the first of the N pointers the *main* pointer which has a distinguished behaviour. The other pointers are refered to as the *auxiliairy* pointers
The element of $\mathtt{S}$ is the state the machine is in. 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 element of $\{0, 1, \ldots, n\}^N$ correspond to the positions of the pointers on the input.

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

Definition 41 ***(transition).***
Let w be a word and $M = (N, \mathtt{S}, \Delta)$ be a pointer machine. A transition *of (w, M) is a triple of configurations*

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

such that

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

Note that the maintenance of $\mathtt{c}''$ as the letter pointed to by the (new) main pointer is not required at the $\mathtt{SWAP}$ phase, which will cause most computation to stop unexpectedly. The point is that it is possible to implement that maintainance using the states of the machine (typically, add a store of the values of the pointers to the state of the machine, which has finitely many configurations), and therefore we favor the simpler and more liberal definition.

Definition 42 ***(acceptation).***
We say that a pointer machine M accepts the word w if any sequence of transitions $\left(C_i \xrightarrow{\mathtt{MOVE}} C'_i \xrightarrow{\mathtt{SWAP}} C''_i\right)$ of (w, M) such that $C''_i = C_{i+1}$ for all i is necessarily finite.
We write $\mathbf{L}(M)$ the set of words accepted by M.

This means informally that we consider that a pointer machine accepts a word if it cannot ever loop, from whatever configuration it starts from. Of course, this rather unusual acceptance condition is given with the acceptance condition of wirings in mind: nilpotency.

Definition 43 ***(machines and wirings).***
Let $M = (N, \mathtt{S}, \Delta)$ *be a pointer machine. We Associate to the* $s \in \mathtt{S}$ *a set of distinct closed terms* $[\mathtt{S}]$ *. We 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*

$$[D] := ([s'] \bullet \mathtt{c}' \bullet \mathtt{d}' \leftharpoonup [s] \bullet \mathtt{c} \bullet \mathtt{d}) \otimes \sigma \ \in [\mathtt{S}] \otimes \Sigma \otimes \mathtt{LR} \otimes \mathcal{S}_n$$

and we define the wiring $[M] \in \mathcal{O}_{\Sigma}^{+}$ *as* $\sum_{D \in \Delta} [D]$ *.*

The translation of definition 42 preserves the language recognized and relates reversibility with isometricity.

Theorem 44 ***(reversibility).***
A pointer machine M *is reversible if and only if* $[M]$ *is an isometric wiring.*

Theorem 45 ***(acceptation).***
For any pointer machine M *,* $\mathbf{L}(M) = \mathbf{L}([M])$ *.*

As pointer machines are expressive enough to solve any (N)LOGSPACE problem, we get the result we aim at.

Theorem 46 ***(space completeness).***
If $L \in$ NLOGSPACE *, then there exist a pointer machine* M *such that* $\mathbf{L}(M) = L$ *.*
Moreover, if $L \in$ LOGSPACE *then* M *can be chosen to be reversible.*

Discussion

With respect to earlier [16,2,27] this work start to clarify one point: what is really needed to carry on the construction that captures logarithmic space computation? Indeed, earlier work used in place of the unification algebra the so-called *hyperfinite factor*, which involved advanced notions of von Neumann algebras theory. Our work shows that the von Neumann structure is indeed not indispensible, whereas the ability to represent the action of permutation groups on an unbounded tensor product seems to be an important point in the relation to pointer machines.

The language of unification gives us a twofold point of view on computation: now that we related wirings and pointer machines, we can start exploring possible extension of our construction, keeping some mathematical structuring in mind. For instance, a natural operation one would like to define on pointer machines would be the one the resets the main pointer to the initial position holding the symbol $\star$. This is not possible within the setting of this article, because of the notion of normative pair: this would require to know in advance the term t_0 corresponding to the initial position.

References

1. Andrea Asperti, Vincent Danos, Cosimo Laneve, and Laurent Regnier. Paths in the lambda-calculus. pages 426–436, 1994.
2. Cl\'e Aubert. Characterizing co-NL by a group action. *CoRR*, abs/1209.3422.
3. Patrick Baillot and Damiano Mazza. Linear logic by levels and bounded time complexity. *Theor. Comput. Sci.*, 411(2):470–503, 2010.
4. Patrick Baillot and Marco Pedicini. Elementary complexity and geometry of interaction. *Fundam. Inform.*, 45(1-2):1–31, 2001.
5. Amir M. Ben-amram. What is a "Pointer machine"? In *Science of Computer Programming*. American Mathematical Society, 1995.
6. Cynthia Dwork, Paris C. Kanellakis, and John C. Mitchell. On the sequential nature of unification. *J. Log. Program.*, 1(1):35–50, 1984.
7. Cynthia Dwork, Paris C. Kanellakis, and Larry J. Stockmeyer. Parallel algorithms for term matching. *SIAM J. Comput.*, 17(4):711–731, 1988.
8. Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of pspace. pages 121–131, 2008.
9. Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
10. Jean-Yves Girard. Geometry of interaction 2: deadlock-free algorithms. pages 76–93, 1988.
11. Jean-Yves Girard. Geometry of interaction 1: Interpretation of system f. In S. Valentini R. Ferro, C. Bonotto and A. Zanardo, editors, *Studies in Logic and the Foundations of Mathematics*, volume Volume 127 of *Logic Colloquium '88 Proceedings of the Colloquium held in Padova*, pages 221–260. Elsevier, 1989.
12. Jean-Yves Girard. Light linear logic. pages 145–176, 1994.
13. Jean-yves Girard. Geometry of interaction III: accommodating the additives. In *Advances in Linear Logic, LNS 222,CUP, 329–389*, page 329–389. Cambridge University Press, 1995.
14. Jean-yves Girard. Proof-nets: The parallel syntax for proof-theory. In *Logic and Algebra*, page 97–124. Marcel Dekker, 1996.
15. Jean-yves Girard. *Geometry of Interaction IV: the Feedback Equation.* 2005.
16. Jean-Yves Girard. Normativity in logic. In *Epistemology versus Ontology*, pages 243–263. 2012.
17. Jacques Herbrand. Recherches sur la théorie de la démonstration. 1930.
18. Martin Hofmann and Sch\"o˙Pointer programs and undirected reachability.
19. Martin Hofmann and Sch\"o˙Pure pointer programs with iteration. *ACM Trans. Comput. Log.*, 11(4).
20. Olivier Laurent. A token machine for full geometry of interaction. pages 283–297, 2001.
21. Harry R. Lewis and Richard Statman. Unifiability is complete for co-NLogSpace. *Inf. Process. Lett.*, 15(5):220–222, 1982.
22. A. Martelli and U. Montanari. *Unification in Linear Time and Space: A Structured Presentation.* Istituto di Elaborazione della Informazione, Consiglio Nazionale delle Ricerche, 1976.
23. Mike Paterson and Mark N. Wegman. Linear unification. *J. Comput. Syst. Sci.*, 16(2):158–167, 1978.
24. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965.
25. Sch\"o˙Space-efficient computation by interaction.
26. Sch\"o˙Stratified bounded affine logic for logarithmic space.
27. Thomas Seiller and Cl\'e Aubert. Logarithmic space and permutations. *CoRR*, abs/1301.3189.