A Language-Theoretical Approach to Descriptive Complexity

Michaël Cadilhac, Andreas Krebs, and Klaus-Jörn Lange

Wilhelm-Schickard-Institut, Universität Tübingen, Sand 13, Tübingen, Germany
michael@cadilhac.name, mail@krebs-net.de, lange@informatik.uni-tuebingen.de

Abstract. Logical formulas are naturally decomposed into their subformulas and circuits into their layers. How are these decompositions expressed in a purely language-theoretical setting? We address that question, and in doing so, introduce a product directly on languages that parallels formula composition. This framework makes an essential use of languages of higher-dimensional words, called *hyperwords*, of arbitrary dimensions. It is shown here that the product thus introduced is associative over classes of languages closed under the product itself; this translates back to extra freedom in the way formulas and circuits can be decomposed.

Keywords: Logic, languages, descriptive complexity, hyperwords, circuits

1 Introduction

The theory of constant-depth polysize unbounded-fan-in circuits (hereafter simply *circuits*) abounds in fine classes of languages and open problems about their relationships. Some of the main classes of focus in the literature are:

- AC^0, the class of languages recognized by circuits with Boolean gates;
- TC^0, based on AC^0 circuits with additional *threshold* gates, which output 1 if the majority of their input bits is 1;
- NC^1, which, while being usually defined with log-depth, polysize, bounded-fan-in Boolean circuits, is also characterized by AC^0 circuits with additional *regular* oracle gates, which output 1 if their input is in a prescribed regular language.

Strikingly, all these classes admit characterizations that rely on language recognition by first-order logic formulas—these are the classical results of [5, 1], that we recall in Proposition 16 (see [10] for a lovely account). In this framework, the variables of a logical formula range over the positions in an input word, and the language described by the formula is the set of words satisfying it. Similarly, algebraic characterizations of AC^0 and NC^1 relying on *programs over finite monoids* [1] and of TC^0 relying on recognition by *typed monoids* [6] are known. This however is not a mere coincidence, and tokens of the pervasiveness of this interplay between logic, circuits, and algebra were unveiled in more general settings [10, 7, 12, 2], including in restrictions of these classes to a linear number

of gates [3]. Each time, these results are shown inductively by identifying building blocks (simple formulas, simple circuits, etc.) and an appropriate composition operation (substitution, stacking of circuits, etc.).

There is, however, a missing link in this picture: a purely language-theoretical construct that would unify these frameworks. As they all are used *in fine* as language specifications, this calls for a better understanding of their building blocks and compositions, without appeal to a specific model of computation. This is what we aim for in this article.

Higher dimensions. A prominent feature of our study is its reliance on words of higher dimension, that we call *hyperwords*. Contrary to previous works where pictures are 2-dimensional, i.e., mappings from $\{1, 2, \dots, m\} \times \{1, 2, \dots, n\}$ to some alphabet [9], our hyperwords are labeled squares, cubes, etc., and more generally, mappings from $\{1, 2, \dots, n\}^d$ to an alphabet. Going to higher dimensions constitutes a severe change that is prompted by multiple considerations: **1.** In the logical framework, composition of formulas (the so-called "substitution") is a process that replaces letter predicates $\mathfrak{c}_a(x)$, asserting that there there is an a at position x in the input, by a formula with one distinguished variable. Generalizing this substitution to a greater number of variables naturally leads to consider letter predicates of the form $\mathfrak{c}_a(x_1, x_2, \dots, x_d)$, hence formulas recognizing d-dimensional hyperwords. **2.** In the circuit framework, one can speak of the language accepted by a circuit with n inputs. However, a *layer* of the circuit may have a polynomial number of input gates, say n^d, and thus accepts a hyperword of dimension d. **3.** Since the early stages of descriptive complexity, there has been a great interest in quantifiers that bind more than one variable. For instance, the *majority of pairs* quantifier, $(\mathrm{Maj}_2\ x, y)[\varphi]$, asserts that there is a majority of positions (i, j) of the input word making $\varphi(x := i, y := j)$ true. Barrington, Immerman, and Straubing conjectured in the seminal paper [1] that Maj_2 is more powerful than the majority quantifier over a single variable, and this was proven in [8]. A quantifier of that type, a so-called *Lindström quantifier*, is entirely described by a set of hyperwords; for instance, the truth value of $(\mathrm{Maj}_2\ x, y)[\varphi]$ depends solely on whether the 2-dimensional hyperword mapping (i, j) to the truth value of $\varphi(x := i, y := j)$ contains a majority of "true." Thus again, quantifiers are determined by hyperword languages.

Our contributions are the following:

1. We adapt the traditional logic framework to the description of hyperword languages, and define a notion of substitution that extends the one for single variable formulas (see Section 4);
2. We introduce a purely language-theoretical framework, relying on hyperword languages, and a product over languages ("block product") that allows to express logic-defined languages (and thus ultimately languages of circuit families) independently of a model (see Sections 3 and 6);
3. We show that the product thus defined verifies a certain associativity property: there is a trade-off between the possible bracketings of an expression and the dimensions of the languages therein (see Theorem 21).

2 Preliminaries

For an integer n, we write $[n]$ for the set $\{1, 2, \ldots, n\}$. For a function $f\colon X \to Y$, and for a set X', we write $f{\restriction_{X'}}$ for the function from $X \cap X'$ to Y that agrees with f on its domain. If $x_1, x_2, \ldots, x_e$ are some variables, we write $\boldsymbol{x}$ for the vector $(x_1, x_2, \ldots, x_e)$, and if $\boldsymbol{i}$ is a vector of same length, then $\boldsymbol{x} = \boldsymbol{i}$ is to be understood component-wise.

In the following, A and B will be alphabets, i.e., finite sets of symbols, and $\mathcal{V}$ will be a finite set of *variable symbols* included in[1] $\{\ldots, \mathfrak{v}_{-2}, \mathfrak{v}_{-1}, \mathfrak{v}_0, \mathfrak{v}_1, \mathfrak{v}_2, \ldots\}$, and we will use $x, y, x_1, x_2, \ldots$ to refer to these variables. Such sets $\mathcal{V}$ are naturally ordered, and we will often speak of the i first variables of $\mathcal{V}$.

A *stripped hyperword* over A of *dimension* $d \geq 0$ and *length* $n \geq 0$ is a map from $[n]^d$ to A; the set of stripped hyperwords of dimension d for any length is written $\mathcal{H}_d(A)$, and in particular, we have that $A^* = \mathcal{H}_1(A)$. We will also naturally identify A with $\mathcal{H}_0(A)$.

Hyperwords will *always* be paired with valuations of a (possibly empty) finite set of variables: we let $\mathcal{H}_d(A) \otimes \mathcal{V}$ be the set of pairs $W = (\mathsf{str}_W, \mathsf{val}_W)$ such that $\mathsf{str}_W \in \mathcal{H}_d(A)$ and $\mathsf{val}_W\colon \mathcal{V} \to \{1, \ldots, n\}$, with n the length of str_W. These objects will be called simply *hyperwords*, and we define the *length* of W, written $|W|$, to be that of str_W, its *strip* to be str_W, and its *valuation* to be val_W. A language of dimension d is then a set of hyperwords of this dimension, and we identify subsets of $\mathcal{H}_d(A)$ with languages in $\mathcal{H}_d(A) \otimes \emptyset$. Further, for a hyperword $W \in \mathcal{H}_d(A) \otimes \mathcal{V}$ and $\boldsymbol{i} \in [|W|]^d$, we write $W(\boldsymbol{i})$ for the letter $\mathsf{str}_W(\boldsymbol{i})$, and if $x \in \mathcal{V}$, then $W(\ldots, x, \ldots)$ denotes $W(\ldots, \mathsf{val}_W(x), \ldots)$. For a variable x that may or may not be in $\mathcal{V}$ and $i \in [|W|]$, we write $W_{x=i}$ for the hyperword with strip str_W and valuation val_W modified so that x is mapped to i (hence x is added to the domain of val_W if $x \notin \mathcal{V}$). Hyperwords of dimension 1 will usually be called *words*. For a language $L \subseteq \mathcal{H}_d(A) \otimes \mathcal{V}$, we denote its characteristic function by $\chi_L\colon \mathcal{H}_d(A) \otimes \mathcal{V} \to \{0, 1\}$.

3 Composing Languages

We begin with an intuitive presentation. Suppose we are given a language $L \subseteq \mathcal{H}_d(A) \otimes \{\mathfrak{v}_1\}$, and we wish to extract from it the language $L' \subseteq \mathcal{H}_d(A)$ of hyperwords in L that have an even valuation of $\mathfrak{v}_1$. In symbols, we want to define $L' = \{W \mid (\exists i \in 2\mathbb{N})[W_{\mathfrak{v}_1=i} \in L]\}$. When checking whether $W \in L'$, we are thus interested in the different values of $\chi_L(W_{\mathfrak{v}_1=i})$ for i ranging from 1 to $|W|$; indeed, if K is the set of words over $\{0, 1\}$ having at least one 1 in an even position, then $W \in L'$ if and only if:

$$\chi_L(W_{\mathfrak{v}_1=1}) \cdot \chi_L(W_{\mathfrak{v}_1=2}) \cdots \chi_L(W_{\mathfrak{v}_1=|W|}) \in K \ .$$

[1] We only make scarce use of the variables with nonpositive indexes explicitly, with the notable exception of the first part of the proof of Theorem 21.

This construction is a particular example of the *block product*[2] of two languages, and we shall later write $L' = K \,\square\, L$. Our definition of the block product follows the definition of Lindström quantifiers (e.g., [1]) by making the following three generalizations:

1. We extend the valuation of $\mathfrak{v}_1$ to a *set* of variables; for instance, for two variables $\mathfrak{v}_1$ and $\mathfrak{v}_2$, rather than checking whether the *word* whose i-th letter is $\chi_L(W_{\mathfrak{v}_1=i})$ belongs to K, it should be checked whether the *hyperword* whose letter at position (i,j) is $\chi_L(W_{\mathfrak{v}_1=i,\mathfrak{v}_2=j})$ belongs to K;
2. The membership tests $\chi_L(W_{\mathfrak{v}_1=i})$ are allowed to range over a finite number of different languages L; this implies that K in our example is not simply a language over $\{0,1\}$, but over $\{0,1\}^k$ for some $k > 0$;
3. We introduce mappings from the truth values of these membership tests to different alphabets; in other words, we implement a mechanism to let K be over any alphabet.

Definition 1 (Simple join). *Let $(L_i)_{i\in[k]}$ be languages. When (and only when) all the L_i's share the same alphabet A, dimension d, and variable set $\mathcal{V}$, we write $\mathcal{L} = [L_1, L_2, \dots, L_k]$ to denote the vector whose i-th component is L_i.*

This vector $\mathcal{L}$ is called a simple join *of length k over $\mathcal{H}_d(A) \otimes \mathcal{V}$, and we naturally extend the characteristic functions to such objects by letting, for any $W \in \mathcal{H}_d(A) \otimes \mathcal{V}$, $\chi_{\mathcal{L}}(W) = \big(\chi_{L_1}(W), \chi_{L_2}(W), \dots, \chi_{L_k}(W)\big) \in \{0,1\}^k$.*

Definition 2 (Block product). *Let K be a language in $\mathcal{H}_e(B) \otimes \mathcal{V}$ and $\mathcal{L}$ be a simple join of length k over $\mathcal{H}_d(A) \otimes (\mathcal{X} \cup \mathcal{V})$ with $\mathcal{X} = \{x_1, x_2, \dots, x_e\}$ the first e variables of $\mathcal{X} \cup \mathcal{V}$, in order. Further, let $g\colon \{0,1\}^k \to B$.*

Let $W \in \mathcal{H}_d(A) \otimes \mathcal{V}$. The transcript *$\tau(W) \in \mathcal{H}_e(B) \otimes \mathcal{V}$ of W is the hyperword with strip:*

$$[|W|]^e \to B$$
$$(i_1, i_2, \dots, i_e) \mapsto g(\chi_{\mathcal{L}}(W_{\boldsymbol{x}=\boldsymbol{i}})) \ ,$$

and valuation val_W. The block product *of K and $\mathcal{L}$ (with* alphabet replacement *g) is then $K \,\square_g\, \mathcal{L} = \{W \in \mathcal{H}_d(A) \otimes \mathcal{V} \mid \tau(W) \in K\}$.*

Notation 3. *We will often use alphabet replacements from $\{0,1\}^k$ to $\{0,1\}$. In this case, we see $0, 1$ as Boolean values, and use the notations $\wedge, \vee, \leftrightarrow, \dots$ directly in the list $\mathcal{L}$. For instance, $L = K \,\square\, [L_1 \vee (L_2 \leftrightarrow L_3)]$ defines $g\colon \{0,1\}^3 \to \{0,1\}$ by $g(i,j,k) = i \vee (j \leftrightarrow k)$, and then $L = K \,\square_g\, [L_1, L_2, L_3]$. Further, we omit the alphabet replacement when it is the identity, and if L is a language, we write $K \,\square\, L$ for $K \,\square\, [L]$.*

The following operators do not directly relate to the block product. However, they will be part of our elementary set of tools to define more complex languages.

[2] This nomenclature stems from the algebraic operation bearing the same name. There is a precise relationship between block products of monoids and block products of languages of words (Definition 2) that will be made explicit in an extended version of this article.

Definition 4 (Variable operators). *Let $L \subseteq \mathcal{H}_d(A) \otimes \mathcal{V}$. The following two operators respectively decrease and increase the number of variables used.*

The variable renaming *identifies and renames variables of $\mathcal{V}$. Let $\sigma\colon \mathcal{V} \to \mathcal{V}'$ be a given partial map, for a set $\mathcal{V}'$ of variables. First, extend σ to all of $\mathcal{V}$ by letting $\sigma(x) = x$ if σ was undefined on x. Then for a valuation val of $\sigma(\mathcal{V})$, write $\sigma^{-1}(\mathsf{val})$ for the valuation of $\mathcal{V}$ mapping x to $\mathsf{val}(\sigma(x))$. The* variable renaming *of L by σ is $\mathsf{ren}(L, \sigma) = \{W \in \mathcal{H}_d(A) \otimes \sigma(\mathcal{V}) \mid (\mathsf{str}_W, \sigma^{-1}(\mathsf{val}_W)) \in L\}$.*

The variable extension *augments the set of variables $\mathcal{V}$ with untested variables. Let $\mathcal{V}'$ be a finite set of variables, the* variable extension *of L by $\mathcal{V}'$ is $\mathsf{var\text{-}ext}(L, \mathcal{V}') = \{W \in \mathcal{H}_d(A) \otimes (\mathcal{V} \cup \mathcal{V}') \mid (\mathsf{str}_W, \mathsf{val}_W{\restriction_{\mathcal{V}}}) \in L\}$.*

4 The Descriptive Complexity Framework

We present a generalized version of the classical framework of descriptive complexity for expressing languages (e.g., [1, 10]). The generalization lies essentially in the ability for a formula to recognize a language of hyperwords. A logic will be given by the set of allowed *quantifiers* and *numerical predicates*, which will have a preset semantics. As an example, we want to be able to write formulas such as $(\mathrm{Maj}_{2,1}\ \mathfrak{v}_1, \mathfrak{v}_2)[\mathfrak{c}_a(\mathfrak{v}_1, \mathfrak{v}_2)]$, expressing that there is a majority of pairs of positions (i, j) such that the 2-dimensional input hyperword has an a in position (i, j). As usual (e.g., [1, 8]), we also allow multiple formulas under the scope of a quantifier.

Definition 5 (Quantifier, numerical predicate). *An* (e, k)-ary quantifier *is a pair (L, g) where $L \subseteq \mathcal{H}_e(A) \otimes \mathcal{V}$, for some alphabet A and variable set $\mathcal{V}$, and $g\colon \{0,1\}^k \to A$. Intuitively, e will be the number of variables quantified and k the number of formulas over which the quantifier ranges.*

An e-ary numerical predicate *is a subset of $\mathcal{H}_1(\{a\}) \otimes \{\mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_e\}$.*

Definition 6 (Logic). *Given a set of quantifiers $\mathcal{Q}$ and a set of numerical predicates $\mathcal{N}$, we define the logic $\mathcal{Q}[\mathcal{N}]$ as the set of following formulas with the provided semantics:*

- Syntax. *A* formula of dimension d over the alphabet A *is built from the following syntax, where the x_i's are variables that are not necessarily distinct, except in Case 3:*

$$\begin{aligned}
\varphi ::= \quad & \mathfrak{c}_a(x_1, x_2, \ldots, x_d) \text{ where } a \in A && (1)\\
\mid\ & N(x_1, x_2, \ldots, x_e) \text{ for any } N \in \mathcal{N} \text{ of arity } e && (2)\\
\mid\ & (Q\ x_1, x_2, \ldots, x_e)[\varphi_1, \varphi_2, \ldots, \varphi_k] \text{ for any } Q \in \mathcal{Q} \text{ of arity } (e, k) && (3)\\
\mid\ & \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi_1 && (4)
\end{aligned}$$

  We rely on the usual vocabulary concerning variables: a variable used in a formula is bounded *if it always appears after being quantified, otherwise it is* free*. This includes the variables that may appear* within *a quantifier, e.g., if $Q = (L, g)$ is a quantifier where $L \subseteq \mathcal{H}_1(A) \otimes \mathcal{V}$, then all variables of $\mathcal{V}$*

are free in $(Q\ x)[\varphi]$. *If* $\{\mathfrak{v}_{i_1}, \mathfrak{v}_{i_2}, \dots, \mathfrak{v}_{i_n}\}$ *are the free variables of* φ, *with* $i_1 < i_2 < \cdots < i_n$, *we write* $\varphi(x_1, x_2, \dots, x_k)$, *with* $k \leq n$, *for the formula* φ *with* $\mathfrak{v}_{i_j}$ *replaced by* x_j, *for all* $j \in [k]$; *we let the formulas obtained in this fashion also belong to* $\mathcal{Q}[\mathcal{N}]$.

– Semantics. *Let* φ *be a formula of dimension* d *over the alphabet* A, *and* $\mathcal{V}$ *a set containing all its free variables. A hyperword* $W \in \mathcal{H}_d(A) \otimes \mathcal{V}$ *is said to be* a model *of* φ, *written* $W \models \varphi$, *when (the cases refer to the above syntax):*
  - *(Case 1)* $W(x_1, x_2, \dots, x_d) = a$, *recalling our use of* $W(\dots, x_i, \dots)$ *as short for* $W(\dots, \mathsf{val}_W(x_i), \dots)$.
  - *(Case 2) The word* $(a^{|W|}, \{\mathfrak{v}_i \mapsto \mathsf{val}_W(x_i)\}_{i \in [e]})$ *is in* N.
  - *(Case 3)* $W' \in L$, *where* $Q = (L, g)$ *with* $L \subseteq \mathcal{H}_e(B) \otimes \mathcal{V}'$, *and* W' *is defined as the hyperword with strip:*

$$[|W|]^e \to B$$
$$(i_1, i_2, \dots, i_e) \mapsto g\big(\ (W_{\boldsymbol{x}=\boldsymbol{i}} \models \varphi_1) \cdot (W_{\boldsymbol{x}=\boldsymbol{i}} \models \varphi_2) \cdots (W_{\boldsymbol{x}=\boldsymbol{i}} \models \varphi_e)\ \big)\ ,$$

    and valuation $\mathsf{val}_W{\restriction_{\mathcal{V}'}}$.
  - *(Case 4) For* $\wedge$, *when* $W \models \varphi_1$ *and* $W \models \varphi_2$, *and likewise for* $\vee$ *and* $\neg$.

Finally, we let $L(\varphi)$, *the language of* φ, *be* $\{W \in \mathcal{H}_d(A) \otimes \mathcal{V} \mid W \models \varphi\}$, *with* $\mathcal{V}$ *the set of free variables of* φ, *and also identify* $\mathcal{Q}[\mathcal{N}]$ *with the class of languages of its formulas.*

Example 7 (Some standard quantifiers). The *first-order quantifiers* $\mathrm{FO} = \{\exists, \forall\}$ are defined as follows. The $(1,1)$-ary quantifier $\exists$ consists of the pair (L, g) where g is the identity over $\{0,1\}$, and L the set of words $\{0,1\}^* \cdot 1 \cdot \{0,1\}^*$. The quantifier $\forall$ is defined similarly with $L = 1^*$.

The (e,k)-ary *majority quantifier* $\mathrm{Maj}_{e,k}$ is the pair (L, g) where $g\colon \{0,1\}^k \to \{-k, \dots, k\}$ computes the difference of the number of 1's and 0's and L consists of hyperwords of $\mathcal{H}_e(\{-k, \dots, k\})$ such that the sum of all letters appearing is greater than 0. The *counting quantifier* $\exists^{=\mathfrak{v}_1}$ can be expressed correspondingly.

Example 8 (Some standard numerical predicates). The 2-ary numerical predicate $=$ is the set of words w such that $\mathsf{val}_w(\mathfrak{v}_1) = \mathsf{val}_w(\mathfrak{v}_2)$; we always assume that this predicate belongs to $\mathcal{N}$ when defining a logic. The 2-ary numerical predicate $<$ is defined similarly. Next, $+$ is a 3-ary numerical predicate for which $\mathsf{val}_w(\mathfrak{v}_1) + \mathsf{val}_w(\mathfrak{v}_2) = \mathsf{val}_w(\mathfrak{v}_3)$. The 2-ary numerical predicate $+1$ is the one for which the words verify $\mathsf{val}_w(\mathfrak{v}_1) + 1 = \mathsf{val}_w(\mathfrak{v}_2)$. The 1-ary numerical predicate max is the set of words w for which $\mathsf{val}_w(\mathfrak{v}_1) = |w|$.

Definition 9 (Substitution). *Let* $\varphi \in \mathcal{Q}[\mathcal{N}]$ *be a formula of dimension* e *over the alphabet* B, *and* $\varphi_1, \varphi_2, \dots, \varphi_k \in \mathcal{Q}[\mathcal{N}]$ *be formulas of dimension* d *over the alphabet* A. *Further let* $g\colon \{0,1\}^k \to B$. *The formula* $\varphi \circ_g [\varphi_1, \varphi_2, \dots, \varphi_k]$ *is obtained from* φ *by replacing its atomic formulas* $\mathfrak{c}_a(x_1, x_2, \dots, x_e)$, $a \in B$, *by:*

$$\bigvee_{\boldsymbol{v} \in g^{-1}(a)} \Big(\bigwedge_{i : v_i = 1} \varphi_i(x_1, x_2, \dots, x_e) \wedge \bigwedge_{i : v_i = 0} \neg \varphi_i(x_1, x_2, \dots, x_e) \Big)\ .$$

This results in a formula of $\mathcal{Q}[\mathcal{N}]$ *of dimension* d *over the alphabet* A *called a* substitution *of* φ.

5 Examples

Example 10 (Existential quantification, logical and). Let $L_i \subseteq \mathcal{H}_d(A) \otimes \{\mathfrak{v}_1\}$, for $i = 1, 2$, be defined as $\{W \mid W \models \varphi_i\}$ for some formulas φ_i of dimension d with free variable $\mathfrak{v}_1$. We wish to express L defined by the formula $(\exists \mathfrak{v}_1)[\varphi_1 \wedge \varphi_2]$ using the block product. To this end, let $E = \{0,1\}^* \cdot 1 \cdot \{0,1\}^*$, we claim that:

$$L = E \,\square\, [L_1 \wedge L_2] \ .$$

Indeed, the transcript of a hyperword W has a 1 in position i iff, by definition, $\chi_{[L_1,L_2]}(W_{\mathfrak{v}_1=i}) = (1,1)$, that is, iff $W_{\mathfrak{v}_1=i} \in L_1 \cap L_2$. The language E then checks that *there exists* one position of the transcript that contains a 1.

Example 11 (Identities). Example 10 seems to indicate that Boolean operations on languages ought to be expressed under the scope of a quantifier (existential in the example). This is correct, but does *not* come at the expense of introducing new variables, since we may speak about *0-dimensional* hyperwords, that is, letters. Thus any language L is equal to $\{1\} \,\square\, L$, where the left-hand side is of dimension 0.

Now let $L \subseteq \mathcal{H}_d(A) \otimes \mathcal{V}$, we wish to express L by using it as the *left-hand side* of a block product. Let $\mathcal{V}' = \{\mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_d\}$, and define for all $a \in A$ the language $C_a \subseteq \mathcal{H}_d(A) \otimes \mathcal{V}'$ to be the set of hyperwords W with $W(\mathfrak{v}) = a$. Finally, with $A = \{a_1, a_2, \ldots, a_\ell\}$, let $g\colon \{0,1\}^\ell \to A$ map $(b_1, b_2, \ldots, b_\ell)$ to a_i if $b_i = 1$ for some unique i—the other values of g being irrelevant. It then holds that:

$$L = L \,\square_g\, [\mathsf{var\text{-}ext}(C_{a_1}, \mathcal{V}), \mathsf{var\text{-}ext}(C_{a_2}, \mathcal{V}), \ldots, \mathsf{var\text{-}ext}(C_{a_\ell}, \mathcal{V})] \ .$$

Example 12 (Boolean operations). Now given a Boolean expression on k variables, that is, a function $g\colon \{0,1\}^k \to \{0,1\}$, and a simple join $[L_1, L_2, \ldots, L_k]$, the language obtained by combining the languages using the expression is:

$$\{1\} \,\square_g\, [L_1, L_2, \ldots, L_k] \ .$$

In particular, we have:

$$L_1 \cup L_2 = \{1\} \,\square\, [L_1 \vee L_2] \ , \quad L_1 \cap L_2 = \{1\} \,\square\, [L_1 \wedge L_2] \ .$$

6 Logics and their Language Classes

In this section, we show that, given a logic, the class of languages recognized by its formulas is the closure, under mainly block product, of a set of languages associated with its quantifiers and numerical predicates.

Definition 13 (Block closure). *A class of languages $\mathcal{C}$ is* block-closed *if it is closed under block products, variable extension, and variable renaming. Further, for a class of languages $\mathcal{C}$, we let $\boxtimes(\mathcal{C})$ be the smallest block-closed class that contains $\mathcal{C}$ and the languages $C_a^{A,d}$, defined for any alphabet A, $a \in A$, and $d \geq 0$, as:*

$$C_a^{A,d} = \{W \in \mathcal{H}_d(A) \otimes \{\mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_d\} \mid W(\mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_d) = a\} \ .$$

For a map $g\colon A \to B$ and a hyperword $W \in \mathcal{H}_d(A) \otimes \mathcal{V}$, write $g(W)$ for the hyperword W where each letter $a \in A$ of str_W is replaced by $g(a)$.

Theorem 14. *Let $\mathcal{Q}$ be a set of quantifiers and $\mathcal{N}$ be a set of numerical predicates. Let $\mathcal{Q}' = \{g^{-1}(L) \mid (L, g) \in \mathcal{Q}\}$. Then $\mathcal{Q}[\mathcal{N}] = \boxplus(\mathcal{Q}' \cup \mathcal{N})$.*

Proof. ($\mathcal{Q}[\mathcal{N}] \subseteq \boxplus(\mathcal{Q}' \cup \mathcal{N})$.) This is proved by induction; let $\varphi \in \mathcal{Q}[\mathcal{N}]$ over A with free variables in $\mathcal{V}$, then:

- If $\varphi \equiv \mathfrak{c}_a(x_1, x_2, \ldots, x_d)$, then $L(\varphi) = \mathsf{ren}(C_a^{A,d}, \sigma)$, with $\sigma = \{\mathfrak{v}_i \mapsto x_i\}_{i\in[e]}$;
- If $\varphi \equiv N(x_1, x_2, \ldots, x_e)$ for $N \in \mathcal{N}$ of arity e, then $L(\varphi) = \mathsf{ren}(N, \sigma)$ with $\sigma = \{\mathfrak{v}_i \mapsto x_i\}_{i\in[e]}$;
- If $\varphi \equiv (Q\ x_1, x_2, \ldots, x_e)[\varphi_1, \varphi_2, \ldots, \varphi_k]$, with $Q = (L, g) \in \mathcal{Q}$ of arity (e, k), then let by induction $L_i = L(\varphi_i) \in \boxplus(\mathcal{Q}' \cup \mathcal{N})$, for $i \in [k]$. Further, rename the variables of all the L_i's and $K = g^{-1}(L)$ so that $x_1, x_2, \ldots, x_e$ appear first among all the variables used, and extend these languages to a common set of variables. Then $L(\varphi) = K \,\square\, [L_1, L_2, \ldots, L_k]$;
- If $\varphi \equiv \varphi_1 \wedge \varphi_2$, then, noting that $\{1\}$, as 0-dimensional, is $C_1^{\{0,1\},0}$, and by Example 12, $L(\varphi) = C_1^{\{0,1\},0} \,\square\, [\mathsf{var\text{-}ext}(L(\varphi_1), \mathcal{V}) \wedge \mathsf{var\text{-}ext}(L(\varphi_2), \mathcal{V})]$;
- The cases $\varphi \equiv \varphi_1 \vee \varphi_2$ and $\varphi \equiv \neg\varphi_1$ are similar to the previous one.

Additionally, renaming of variables is achieved through ren. In each case, we inductively have that $L(\varphi) \in \boxplus(\mathcal{Q}' \cup \mathcal{N})$.

($\boxplus(\mathcal{Q}' \cup \mathcal{N}) \subseteq \mathcal{Q}[\mathcal{N}]$.) Again, this is done by induction; let $L \in \boxplus(\mathcal{Q}' \cup \mathcal{N})$, with $L \subseteq \mathcal{H}_d(A) \otimes \mathcal{V}$, then:

- If $L = N$ for $N \in \mathcal{N}$ of arity e, then $L = L(\varphi)$ for $\varphi \equiv N(\mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_e)$ seen as a formula of dimension 1 over $\{a\}$;
- If $L = g^{-1}(L')$ for $Q = (L', g) \in \mathcal{Q}$, then $A = \{0,1\}^k$ for some k. We then have that $L = L(\varphi)$ with:

$$\varphi \equiv (Q\ \mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_d)\Big[\bigvee_{\boldsymbol{u}\in\{0,1\}^k : u_1 = 1} \mathfrak{c}_{\boldsymbol{u}}(\mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_d), \quad \vdots \quad \bigvee_{\boldsymbol{u}\in\{0,1\}^k : u_k = 1} \mathfrak{c}_{\boldsymbol{u}}(\mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_d)\Big]\ ;$$

- If $L = C_a^{A,d}$, then $L = L(\varphi)$ with $\varphi \equiv \mathfrak{c}_a(\mathfrak{v}_1, \mathfrak{v}_2, \ldots, \mathfrak{v}_d)$ seen as a formula over A;
- If $L = \mathsf{var\text{-}ext}(L', \mathcal{V}')$, then with φ' such that $L' = L(\varphi')$, define φ as the formula $\varphi' \wedge \bigwedge_{x\in\mathcal{V}'} x = x$. We thus have that $L(\varphi)$ is $L(\varphi')$ over the variables $\mathcal{V} \cup \mathcal{V}'$, hence $L = L(\varphi)$;
- If $L = \mathsf{ren}(L', \sigma)$, then we simply apply the renaming σ to the formula defining L';
- Finally, if $L = K \,\square_g\, [L_1, L_2, \ldots, L_k]$, let φ_i such that $L(\varphi_i) = L_i$ for all i, and φ_K such that $L(\varphi_K) = K$, then $L = L(\varphi_K \circ_g [\varphi_1, \varphi_2, \ldots, \varphi_k])$. ◻

A salient property of this characterization is that there is no syntactic difference made between the languages coming from quantifiers, and those coming from

numerical predicates. From this, we naturally derive the following restatement of Theorem 14 starting from languages:

Theorem 15. *Let $\mathcal{C}$ be a class of languages containing the numerical predicate $=$. Let $\mathcal{Q}$ be the set of quantifiers (L, g) such that $L \in \mathcal{C}$. It holds that $\mathcal{Q}[=] = \boxtimes(\mathcal{C})$.*

We note that Theorem 14 immediately implies that some complexity classes can be expressed as the block-closure of simple languages, namely:

Proposition 16. *The following equalities hold:*
- *DLOGTIME-uniform* $\mathsf{TC}^0 = \boxtimes(\{\mathrm{Maj}_{2,1}, <\})$*;*
- *DLOGTIME-uniform* $\mathsf{NC}^1 = \circledast(\{\mathrm{Maj}_{2,1}, S_5, <\})$*, with S_5 the symmetric group on 5 elements, seen as the language of words $\sigma_1\sigma_2\cdots\sigma_n$, with each $\sigma_i \in S_5$, that evaluate to the identity permutation;*
- $\mathsf{P} = \boxtimes(\{\mathrm{Maj}_{2,1}, \mathrm{CVP}, <\})$*, where* CVP *is the* circuit valuation problem*, that is, encoding of Boolean circuits that evaluate to one.*

7 Associativity of the Block Product

In the context of the block product of algebraic structures,[3] it is well known that parenthesizing plays a crucial role. Indeed, the composition $(M \,\square\, N) \,\square\, K$ is sometimes called the *weak* product [11, 3], by opposition to the *strong* one $M \,\square\, (N \,\square\, K)$, and it can be proved that the former recognizes, in general, less languages than the latter. Similarly—equivalently in fact [11, 12]—the classical notion of formula substitution (akin to our definition but with formulas of dimension one) depends intrinsically on the parenthesizing: $\varphi_1 \circ (\varphi_2 \circ (\varphi_3 \circ \cdots))$ can express all formulas starting from formulas of depth 1 (i.e., formulas with one quantifier), while $((\cdots(\varphi_1 \circ \varphi_2) \circ \varphi_3) \circ \cdots)$ can only express formulas with two variables (that may be reused). Here, we show that we can get more freedom in the parenthesizing, provided that we allow products of languages of higher dimensions. We place this result in a purely language-theoretical framework (i.e., with languages and block products), and by Theorem 14 and its proof, it would carry over to the logical setting (i.e., with logical formulas and substitutions).

Naturally, as products of one-dimensional languages are nonassociative, we cannot hope for $K \,\square\, (L_1 \,\square\, L_2)$ to be equal to $(K \,\square\, L_1) \,\square\, L_2$ in general. We will however see in the proof of the forthcoming Theorem 21, that it is enough to provide a *dimensional jump* of L_1:

Definition 17 (Dimensional jump). *Let $L \subseteq \mathcal{H}_d(A) \otimes \mathcal{V}$. For $0 < c \leq |\mathcal{V}|$, we let $[\![L]\!]^c$, the c-dimensional jump of L, be the language of hyperwords W in $\mathcal{H}_{c+d}(A) \otimes \mathcal{V}$ defined as copies of L in the following sense. Let $\{x_1, x_2, \ldots, x_c\}$ be the c first variables of $\mathcal{V}$. For $\boldsymbol{v} \in [|W|]^c$, define $W(\boldsymbol{v}, \bullet)$ as the d-dimensional hyperword of strip mapping $\boldsymbol{u} \in [|W|]^d$ to $W(\boldsymbol{v}, \boldsymbol{u})$, and of valuation val_W. Then:*

$$W \in [\![L]\!]^c \quad \Leftrightarrow \quad W(x_1, x_2, \ldots, x_c, \bullet) \in L \ .$$

[3] The reader not versed in that topic can think of block products of monoids as block products of the languages of dimension 1 recognized by them.

If $[L_1, \dots, L_k]$ *is a join, we let* $[\![L_1, \dots, L_k]\!]^c = [[\![L_1]\!]^c, \dots, [\![L_k]\!]^c]$.

Further, to treat simple lists, we will need the following symmetric operators that increase the dimension of hyperwords by a constant, the original hyperwords appearing in the first or the last components. With the notations of Definition 17:

Definition 18 (Dimensional extensions). *The* right dimensional extension *of* $L \subseteq \mathcal{H}_d(A) \otimes \mathcal{V}$ *for any* $e > 0$*, written* $\mathsf{dim\text{-}ext}(L, e)$*, is defined as the set* $\{W \in \mathcal{H}_{d+e}(A) \otimes \mathcal{V} \mid (\forall \boldsymbol{v} \in [|W|]^e)[W(\boldsymbol{v}, \bullet) \in L]\}$.

Similarly, its left dimensional extension $\mathsf{dim\text{-}ext}(e, L)$ *is the set of hyperwords* $\{W \in \mathcal{H}_{e+d}(A) \otimes \mathcal{V} \mid (\forall \boldsymbol{v} \in [|W|]^e)[W(\bullet, \boldsymbol{v}) \in L]\}$.

Finally, we will need to be able to "enlarge" the alphabets at hand:

Definition 19 (Alphabet product extension). *Let* $L \subseteq \mathcal{H}_d(A) \otimes \mathcal{V}$ *and* B *be an alphabet. The* right alphabet product extension *of* L *by* B*, written* $\mathsf{alph\text{-}prod}(L, B)$*, is the set of hyperwords in* $\mathcal{H}_d(A \times B) \otimes \mathcal{V}$ *such that dropping the second component of each letter gives a hyperword in* L*. The* left alphabet product extension $\mathsf{alph\text{-}prod}(B, L)$ *is defined symmetrically, resulting in hyperwords in* $\mathcal{H}_d(B \times A) \otimes \mathcal{V}$.

Lemma 20. *Any block-closed class* $\boxtimes(\mathcal{C})$ *containing the language* 1^* *is closed under dimensional jump, dimensional extensions, and alphabet product extensions.*

The aforementioned associativity property of the block product is then:

Theorem 21. *Every language of a block-closed class* $\boxtimes(\mathcal{C})$ *can be written from the languages of* $\mathcal{C}$ *and the languages* $C_a^{A,d}$ *using block products, variable extensions, variable renaming, dimensional jump and extensions, and alphabet product extensions, in such a way that no right-hand side of a block product contains a block product.*

Proof. Any language of $\boxtimes(\mathcal{C})$ can be written, by definition, from the languages of $\mathcal{C}$ and the languages $C_a^{A,d}$ using block products, variable extension, and variable renaming. It is not hard to show that the variable related operators and the dimensional jump can be pushed to the language level, so that a block product is never under the scope of such operators.

To show the main claim, we proceed inductively on the structure of the expression defining a language L of $\boxtimes(\mathcal{C})$, assuming that the variable operators are at the language level. The claim is true for languages of $\mathcal{C}$, their jumps, and their variable extensions and renaming.

We consider first a simplified situation. Let K, L_1, L_2 be languages of dimensions i, j, and k respectively. We claim that $K \,\square\, (L_1 \,\square\, L_2) = (K \,\square\, [\![L_1]\!]^i) \,\square\, L_2$, assuming that the left-hand side is well-defined.

Indeed, let W be a hyperword; we show that the transcript of W at the outermost product of the left-hand side is the same as the transcript of W at the innermost product of the right-hand side. This proves the equality, as the membership of W to either side depends only on this transcript.

The transcript W' of W at the outermost product of the left-hand side is the i-dimensional hyperword whose strip maps $\boldsymbol{v}$ to 1 iff $W'' = W_{\boldsymbol{x}=\boldsymbol{v}} \in L_1 \square L_2$, where $\boldsymbol{x}$ denotes the i first variables of L_1. In turn, this holds iff the transcript of W'' at the innermost product of the left-hand side is in L_1; define U as the $(i+j)$-dimensional hyperword such that $U(\boldsymbol{v}, \bullet)$ is that transcript, for any $\boldsymbol{v}$ of dimension i, and valuation val_W. We have that $W'(\boldsymbol{v}) = 1$ iff $U(\boldsymbol{v}, \bullet)_{\boldsymbol{x}=\boldsymbol{v}} \in L_1$, that is, iff $U_{\boldsymbol{x}=\boldsymbol{v}} \in [\![L_2]\!]^i$. Now U is precisely the transcript of W at the outermost product of the *right*-hand side. Thus the transcript of U at the innermost product of the right-hand side is an i-dimensional hyperword whose strip maps $\boldsymbol{v}$ to 1 iff $U_{\boldsymbol{x}=\boldsymbol{v}} \in [\![L_2]\!]^i$, and this transcript is W'. This shows the equality.

We now introduce simple lists in two steps. Writing $[L_1, L_2] \square_g \mathcal{L}$ for the simple list $[L_1 \square_g \mathcal{L}, L_2 \square_g \mathcal{L}]$, first note that:

$$K \square_f \left([L_1, L_2] \square_g \mathcal{L}\right) = (K \square_f [\![L_1, L_2]\!]^i) \square_g \mathcal{L} \ .$$

Now to treat the general case and conclude this proof, consider the expression $K \square_f [L_1 \square_g \mathcal{L}, L_2 \square_{g'} \mathcal{L}']$. Clearly, for it to be well-defined, L_1 and L_2 must have the same set of variables, thus write $L_i \subseteq \mathcal{H}_{d_i}(A_i) \otimes \mathcal{V}$, $i = 1, 2$. Further, define $L_1' = \mathsf{alph\text{-}prod}(L_1, A_2)$ and $L_2' = \mathsf{alph\text{-}prod}(A_1, L_2)$. Using techniques similar to the above, we may assume that all the languages in $\mathcal{L}$ and $\mathcal{L}'$ are over the variables $\mathcal{X} \uplus \mathcal{V}$ and $\mathcal{X}' \uplus \mathcal{V}$, respectively, so that: 1. $|\mathcal{X}| = d_1$, $|\mathcal{X}'| = d_2$; 2. All the variables in $\mathcal{X}$ are smaller than those in $\mathcal{X}'$; and 3. All the variables in $\mathcal{X}'$ are smaller than those in $\mathcal{V}$. Finally, write $g''(\boldsymbol{u}, \boldsymbol{v}) = (g(\boldsymbol{u}), g(\boldsymbol{v}))$. It then readily holds that the above expression is equal to:

$$K \square_f \left(\left[\mathsf{dim\text{-}ext}(L_1, d_2), \mathsf{dim\text{-}ext}(d_1, L_2) \right] \square_{g''} \left[\mathsf{var\text{-}ext}(\mathcal{L}, \mathcal{X}'), \mathsf{var\text{-}ext}(\mathcal{L}', \mathcal{X}) \right] \right) \ ,$$

where $\mathsf{var\text{-}ext}$ is applied component-wise to all languages of $\mathcal{L}$ and $\mathcal{L}'$. This concludes the proof, as this is of the simpler above form. □

Example 22 (Majorities). As already alluded to, the majority of pairs quantifier, $\mathrm{Maj}_{2,1}$, is in general more powerful than the simple majority quantifier, $\mathrm{Maj}_{1,1}$, even when the latter is nested. Thus, it is interesting to see what kind of quantifiers arises from the parenthesizing given by Theorem 21.

Consider the language M of words over $\{0, 1\}$ containing more 1's than 0's. Let $L' = M \square (\mathsf{var\text{-}ext}(M, \{\mathfrak{v}_1\}) \square L)$ be a well-defined language, where $\mathfrak{v}_1$ is the first variable of L. Then $L' = (M \square [\![\mathsf{var\text{-}ext}(M, \{\mathfrak{v}_1\})]\!]^1) \square L$, by the proof of Theorem 21. Let $Z = (M \square [\![\mathsf{var\text{-}ext}(M, \{\mathfrak{v}_1\})]\!]^1)$, which is a subset of $\mathcal{H}_2(\{0, 1\})$; we describe Z. A hyperword $W \in \mathcal{H}_2(\{0, 1\})$ is in Z iff its transcript is in M, by definition. This transcript has a 1 in position $i \in [|W|]$ iff $W(i, \bullet) \in M$. Thus, seeing two-dimensional hyperwords as arrays, a hyperword W is in Z iff there is a majority of rows of W that contain a majority of 1. There lies the intrinsic difference with $\mathrm{Maj}_{2,1}$, a quantifier that would translate to a language of two-dimensional hyperwords having more 1's than 0's.

For two block-closed classes $\mathcal{C}$ and $\mathcal{D}$, write $\mathcal{C} \square \mathcal{D}$ for the smallest block-closed class containing the languages $L \square \mathcal{L}'$ for all $L \in \mathcal{C}$ and $\mathcal{L}'$ a simple join of languages in $\mathcal{D}$. We then have for any block-closed classes $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$: $\square(\mathcal{D} \square \mathcal{E}) = (\mathcal{C} \square \mathcal{D})\square$.

8 Conclusion

We presented a novel purely language-theoretical framework to express classes of languages described by logics. This addresses two shortcomings of the similar *algebraic* theory of typed monoids [6, 7]. First, quantifiers on tuples can be expressed, providing for instance a shorter, arguably more compelling characterization of TC^0, and thus overcoming the limitation of "linear fan-in." Second, by allowing words of higher dimensions, we obtain a product mimicking the classical *block product* of algebraic structures that exhibits a property reminiscent of associativity—this may allow to translate techniques than only applied to weak parenthesizing (e.g., [4]) to a more general setting.

We believe that the results herein advocate for the use of hyperwords, leading to a unified framework in which the freedom of speaking of partial formulas (and hence partial circuits) is balanced by the dimensions used in expressing their composition.

Acknowledgments. The first author thanks Charles Paperman for stimulating discussions.

References

1. Barrington, D.A.M., Immerman, N., Straubing, H.: On uniformity within NC^1. J. Comput. Syst. Sci 41(3), 274–306 (1990)
2. Behle, C., Lange, K.J.: FO[<]-Uniformity. In: Proc. 21st Annual IEEE Conference on Computational Complexity (CCC'06). pp. 183 – 189 (2006)
3. Behle, C., Krebs, A., Mercer, M.: Linear circuits, two-variable logic and weakly blocked monoids. In: Mathematical Foundations of Computer Science. LNCS, vol. 4708, pp. 147–158. Springer-Verlag (2007)
4. Behle, C., Krebs, A., Reifferscheid, S.: Regular languages definable by majority quantifiers with two variables. In: Developments in Language Theory. pp. 91–102. LNCS, Springer (2009)
5. Immerman, N.: Expressibility and Parallel Complexity. SIAM Journal on Computing 18(3), 625–638 (1989)
6. Krebs, A., Lange, K.J., Reifferscheid, S.: Characterizing TC^0 in terms of infinite groups. Theory Comput. Syst. 40(4), 303–325 (2007)
7. Krebs, A.: Typed semigroups, majority logic, and threshold circuits. Ph.D. thesis, Eberhard Karls University of Tübingen (2008)
8. Lautemann, C., McKenzie, P., Schwentick, T., Vollmer, H.: The descriptive complexity approach to LOGCFL. J. Comput. Syst. Sci. 62(4), 629–652 (2001)
9. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Vol. 3: Beyond Words. Springer-Verlag New York, Inc., New York, NY, USA (1997)
10. Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Birkhäuser, Boston (1994)
11. Straubing, H., Thérien, D.: Weakly Iterated Block Products of Finite Monoids. In: LATIN 2002: Theoretical Informatics. pp. 91–104. LNCS, Springer (2002)
12. Thérien, D., Wilke, T.: Nesting until and since in linear temporal logic. Theory Comput. Syst. 37(1), 111–131 (2004)