Restricted Two-Variable FO+MOD Sentences, Circuits and Communication Complexity

Pascal Tesson[1,⋆] and Denis Thérien[2,⋆⋆]

[1] Département d'Informatique et de Génie Logiciel, Université Laval
pascal.tesson@ift.ulaval.ca
[2] School of Computer Science, McGill University
denis@cs.mcgill.ca

Abstract. We obtain a logical characterization of an important class of regular languages, denoted $\mathcal{DO}$, and of its most important subclasses in terms of two-variable sentences with ordinary and modular quantifiers but in which all modular quantifiers lie outside the scope of ordinary quantifiers. The result stems from a new decomposition of the variety of monoids **DO** in terms of iterated block products.

This decomposition and the ensuing logical characterization allows us to shed new light on recent results on regular languages which are recognized by bounded-depth circuits with a linear number of wires and regular languages with small communication complexity.

1 Introduction

Descriptive complexity uses logical sentences to define languages. For instance, one can view the sentence

$$\exists x \exists y \exists z (x < y < z \wedge Q_a x \wedge Q_b y \wedge Q_c z)$$

as defining the set of words in which there are positions x, y, z with $x < y < z$ holding the letters a, b and c respectively, i.e. the set $\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*$. A large amount of research has sought to understand the expressive power of such sentences. In particular, the celebrated result of McNaughton and Papert [6] shows that languages definable by a first-order sentence which, as the one above, use only the order predicate $<$ are exactly the star-free regular languages. By a result of Schützenberger, these are also the regular languages recognizable by an aperiodic (or group-free) monoid and we can thus decide whether a language is **FO** definable. Kamp [4] further showed that these **FO** sentences could be restricted to use only three variables (provided that they can be reused). Much later, Thérien and Wilke characterized languages definable by **FO** formulas using only *two* variables ($\mathbf{FO_2}$) [16] as languages recognized by monoids in **DA**.

⋆ Part of this research took place while the author was at the University of Tübingen, supported by the von Humboldt Foundation.
⋆⋆ Research supported in part by NSERC, FQRNT and the von Humboldt Foundation.

L. Caires et al. (Eds.): ICALP 2005, LNCS 3580, pp. 526–538, 2005.

When **FO** is augmented with modular quantifiers (checking whether a property holds for $i \pmod p$ of the positions in the string), all languages recognizable by solvable monoids can further be described. In this case too, three variables suffice to obtain full expressivity. The power of $\mathbf{FO}+\mathbf{MOD}$ two-variable sentences was studied in [11, 12] and the results depended crucially on showing that every such sentence was equivalent to one where no ordinary quantifier occurs within the scope of a modular quantifier. In this paper, we handle the other extreme case, that is sentences in which no modular quantifier lies in the scope of an ordinary quantifier. This restriction is meaningful since we show that these are provably less expressive than general two-variable sentences and can provide a logical characterization for important subclasses of regular languages.

The key to our new results is a decomposition of the variety of monoids **DO** (precise definitions will be given in Section 2) in terms of weakly-iterated block products, an idea put forward in [12]. That result is interesting in its own right but our motivation comes from two recent results in complexity theory. We show that a regular language is definable by a two-variable sentence in which no modular quantifier lies in the scope of another quantifier iff it is recognized by a monoid in the class $\mathbf{DO} \cap \overline{\mathbf{Ab}}$. This happens to be *precisely* the class of regular languages that can be recognized by ACC^0-circuits with a linear number of wires [5] and by $O(\log n)$-cost communication protocols [15]. In contrast, languages defined by a two-variable sentence without this restriction can be computed by ACC^0-circuits using a linear number of *gates* and we believe that they are the only regular languages with this property. The logical characterization of languages recognized by monoids in $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ sheds new light on the circuit complexity and communication complexity upper bounds, highlighting the interplay between logic, algebra, complexity and formal languages in this context.

In Section 2, we introduce the algebraic tools needed for our discussion. In Section 3, we decompose subvarieties of **DO** using weakly iterated block-products and use this to obtain a logical characterization of the corresponding regular languages in Section 4. Finally, we discuss the applications of these results to circuit complexity and communication complexity in Section 5.

2 Finite Monoids and Regular Languages

We sketch here the algebraic background needed for our discussion. For a more thorough overview of algebraic automata theory, we refer the reader to e.g. [7]. A *monoid* M is a set with a binary associative operation (which we denote multiplicatively) and a distinguished identity element 1_M. For an alphabet Σ, the set of finite words Σ^* forms a monoid under concatenation and with identity element ϵ. For a finite monoid M and language $K \subseteq \Sigma^*$, we say that K is recognized by M if there exists a homomorphism $h : \Sigma^* \to M$ and a subset $T \subseteq M$ such that $K = h^{-1}(T)$. A simple variant of Kleene's Theorem states that a language is regular iff it can be recognized by some finite monoid. In fact, all monoids considered in this paper, with the exception of the free monoid Σ^*, will be finite. It is useful to point out that if K and K' are recognized by M and

M' respectively then K's complement is recognized by M and both $K \cup K'$ and $K \cap K'$ are recognized by $M \times M'$.

A class of finite monoids is a *variety* if it is closed under direct product, homomorphic images and submonoids. In particular, we will deal with the variety of solvable groups $\mathbf{G_{sol}}$, p-groups $\mathbf{G_p}$, Abelian groups $\mathbf{Ab}$, Abelian p-groups $\mathbf{Ab_p}$, aperiodic or group-free monoids $\mathbf{A}$, semilattices $\mathbf{SL}$, i.e. monoids satisfying $x^2 = x$ and $xy = yx$, and the variety $\mathbf{DO}$ of monoids M which satisfy the identity $(xy)^\omega (yx)^\omega (xy)^\omega = (xy)^\omega$ for all $x, y \in M$. In fact, ω can be chosen as the smallest integer such that $x^{2\omega} = x^\omega$ for all $x \in M$.

Furthermore, for any variety of groups $\mathbf{H}$, we denote as $\overline{\mathbf{H}}$ the variety of monoids whose subgroups all lie in $\mathbf{H}$. For any variety $\mathbf{V}$ of monoids, we will denote as $L(\mathbf{V})$ the class of languages that can be recognized by a monoid of $\mathbf{V}$. These *varieties of languages* are fairly robust and in particular are closed under Boolean operations. We focus on varieties of the form $\mathbf{DO} \cap \overline{\mathbf{H}}$ and we give here a description of the corresponding regular languages.

We say that words $x, y \in \Sigma^*$ are M-equivalent if $h(x) = h(y)$ for any homomorphism $h : \Sigma^* \to M$. Let $\alpha(x) \subseteq \Sigma$ be the set of letters occurring in x. For $a \in \alpha(x)$, the *a-left decomposition* of x is the factorization $x = x_0 a x_1$ with $a \notin \alpha(x_0)$. The *a-right decomposition* is defined symmetrically. For a finite group G we define the congruence $\sim^G_{n,k}$ on Σ^* with $|\Sigma| = n$ by induction on $n + k$. First, $x \sim^G_{n,0} y$ iff x, y are G-equivalent. Next, we let $x \sim^G_{n,k} y$ iff:

1. $x \sim^G_{n,k-1} y$;
2. $\alpha(x) = \alpha(y)$;
3. For any $a \in \alpha(x) = \alpha(y)$, if $x = x_0 a x_1$ and $y = y_0 a y_1$ are the a-left decompositions of x and y then $x_0 \sim^G_{n-1,k} y_0$ and $x_1 \sim^G_{n,k-1} y_1$;
4. For any $a \in \alpha(x) = \alpha(y)$, if $x = x_0 a x_1$ and $y = y_0 a y_1$ are the a-right decompositions of x and y then $x_0 \sim^G_{n,k-1} y_0$ and $x_1 \sim^G_{n-1,k} y_1$.

This equivalence relation is well-defined since $|\alpha(x_0)| < |\alpha(x)|$ in (3) and $|\alpha(x_1)| < |\alpha(x)|$ in (4). These congruences allow us to describe languages in $L(\mathbf{DO} \cap \overline{\mathbf{H}})$ and this will be crucial in the proof of Theorem 3.

Theorem 1 ([15]). *Let K be a language in Σ^* with $|\Sigma| = n$. Then K is in $L(\mathbf{DO} \cap \overline{\mathbf{H}})$ iff K is the union of $\sim^G_{n,k}$-classes for some $k \in \mathbb{N}$ and $G \in \mathbf{H}$.*

The following simple fact will also prove useful for Theorem 3 and Lemma 4:

Lemma 1 (e.g. [7]). *A language $K \in \Sigma^*$ can be recognized by a semilatice iff it is in the Boolean algebra generated by the languages $\Sigma^* a \Sigma^*$. Furthermore, u, v are M-equivalent for any semilattice M iff $\alpha(u) = \alpha(v)$.*

Let M and N be finite monoids. As in [11] we denote the operation of M as $+$ and its identity element as 0 to distinguish it from the operation of N. A left-action of N on M is a function mapping pairs $(n, m) \in N \times M$ to $nm \in M$ and satisfying $n(m_1 + m_2) = nm_1 + nm_2$, $n_1(n_2 m) = (n_1 n_2)m$, $n0 = 0$ and $1m = m$. Right actions are defined symmetrically. If we have both a right and a left-action of N on M that further satisfy $n_1(mn_2) = (n_1 m)n_2$, we can construct the

bilateral semidirect product $M**N$ which we define as the monoid with elements in $M \times N$ and multiplication defined as $(m_1, n_1)(m_2 n_2) = (m_1 n_2 + n_1 m_2, n_1 n_2)$. This operation is associative and $(0, 1)$ acts as an identity for it.

For varieties $\mathbf{V}, \mathbf{W}$, we denote $\mathbf{V}\Box\mathbf{W}$ the variety generated by all semidirect products $M * *N$ with $M \in \mathbf{V}$, $N \in \mathbf{W}$. For varieties $\mathbf{U}, \mathbf{V}, \mathbf{W}$ we always have $(\mathbf{U}\Box\mathbf{V})\Box\mathbf{W} \subseteq \mathbf{U}\Box(\mathbf{V}\Box\mathbf{W})$ but the containment is strict in general. Block-product decompositions of varieties traditionally use the stronger bracketing but [11] showed the relevance of the weak bracketing, particularly in relation to two-variable logical sentences.

The languages recognized by $\mathbf{V}\Box\mathbf{W}$ can be conveniently described in terms of languages recognized by $\mathbf{V}$ and $\mathbf{W}$. For a monoid $N \in \mathbf{W}$, an *N-transduction* τ is a function determined by two homomorphisms $h_l, h_r : \Sigma^* \to N$ and mapping words in Σ^* to words in $(N \times \Sigma \times N)^*$. For a word $w = w_1 \dots w_n \in \Sigma^*$ we set $\tau(w) = \tau(w_1)\tau(w_2)\dots\tau(w_n)$ with $\tau(w_i) = (h_l(w_1 \dots w_{i-1}), w_i, h_r(w_{i+1} \dots w_n))$. For a language $K \subseteq (N \times \Sigma \times N)^*$, let $\tau^{-1}(K) = \{w \in \Sigma^* : \tau(w) \in K\}$.

Theorem 2. *[9, 7] A regular language lies in $L(\mathbf{V}\Box\mathbf{W})$ iff it is the Boolean combination of languages in $L(\mathbf{W})$ and languages $\tau^{-1}(K)$ for some $K \in \mathbf{V}$ and N-transduction τ with $N \in \mathbf{W}$.*

A *pointed word* is a pair (w, p) consisting of a word $w \in \Sigma^*$ and a pointer p with $1 \leq p \leq |w|$. A *pointed language* $\dot{K}$ is a set of such structures and, as in [17], we extend the notion of monoid recognizability to these languages: $\dot{K}$ is recognized by M if there are homomorphisms $h_l, h_r : \Sigma^* \to M$ and a set of triples $T \subseteq (M \times \Sigma \times M)$ such that $\dot{K} = \{(w, p) : (h_l(w_1 \dots w_{p-1}), w_p, h_r(w_{p+1} \dots w_{|w|})) \in T\}$. For a variety $\mathbf{V}$ we will denote as $P(\mathbf{V})$ the set of pointed languages recognized by a monoid in $\mathbf{V}$. Abusing our terminology, it will be convenient to think of ordinary words in Σ^* as pointed words with $p = 0$ and thus view $L(\mathbf{V})$ as a subset of $P(\mathbf{V})$.

3 A Weak Block Product Decomposition of DO

In this section, we characterize each variety $\mathbf{DO} \cap \overline{\mathbf{H}}$ for a variety of groups $\mathbf{H}$ using weakly iterated block products. The idea is similar to that of [11] but we also need the combinatorial description of the corresponding regular languages.

Theorem 3. *Let $\mathbf{H}$ be a variety of finite groups, and $\mathbf{V_H}$ be the smallest variety such that $\mathbf{H} \subseteq \mathbf{V_H}$ and $\mathbf{V_H}\Box\mathbf{SL} \subseteq \mathbf{V_H}$, then $\mathbf{V_H} = \mathbf{DO} \cap \overline{\mathbf{H}}$.*

Proof. Clearly, $\mathbf{DO} \cap \overline{\mathbf{H}}$ contains $\mathbf{H}$ and we also need to show that it is closed under block product with $\mathbf{SL}$. First, we claim that $\mathbf{DO}\Box\mathbf{SL} = \mathbf{DO}$. Let $M \in \mathbf{DO}$ and $N \in \mathbf{SL}$. As we did earlier, we denote the multiplication of M additively (even though M is not necessarily commutative). Since $M \in \mathbf{DO}$ there exists an integer k such that $k(v + w) + k(w + v) + k(v + w) = k(v + w)$ and $(2k)v = kv$ for all $v, w \in M$ and we prove that any bilateral semidirect product $M * *N$ satisfies the identity $(xy)^{2k}(yx)^{2k}(xy)^{2k} = (xy)^{2k}$.

Let $x = (m_1, n_1)$ and $y = (m_2, n_2)$ be arbitrary elements of $M * *N$. By definition of the bilateral semidirect product, we have $xy = (m_1n_2+n_1m_2, n_1n_2)$ and so $(xy)^{2k} = (z, (n_1n_2)^{2k})$ where

$$z = m_1n_2(n_1n_2)^{2k-1} + n_1m_2(n_1n_2)^{2k-1} + \ldots + (n_1n_2)^{2k-1}n_1m_2$$

Since N is commutative and idempotent ($n^2 = n$), this is simply:

$$\begin{aligned} z &= m_1(n_1n_2) + n_1m_2(n_1n_2) + \ldots + (n_1n_2)m_2 \\ &= m_1(n_1n_2) + n_1m_2(n_1n_2) + (2k-1)\left[(n_1n_2)m_1(n_1n_2) + (n_1n_2)m_2(n_1n_2)\right] \\ &\quad +(n_1n_2)m_2n_1 + (n_1n_2)m_2 \end{aligned}$$

By the same argument $(xy)^{2k}(yx)^{2k}(xy)^{2k}$ is the pair (z', n_1n_2) with

$$\begin{aligned} z' = {} & m_1(n_1n_2) + n_1m_2(n_1n_2) + (2k-1)\left[(n_1n_2)m_1(n_1n_2) + (n_1n_2)m_2(n_1n_2)\right] \\ & +(2k)[(n_1n_2)m_2(n_1n_2) + (n_1n_2)m_1(n_1n_2)] \\ & +(2k-1)\left[(n_1n_2)m_1(n_1n_2) + (n_1n_2)m_2(n_1n_2)\right] + (n_1n_2)m_2n_1 + (n_1n_2)m_2. \end{aligned}$$

If we let $v = (n_1n_2)m_1(n_1n_2)$ and $w = (n_1n_2)m_2(n_1n_2)$, then the middle part of the product is simply $(2k-1)(v+w)+2k(w+v)+(2k-1)(v+w)$ and since M satisfies $k(v+w)+k(w+v)+k(v+w) = k(v+w)$ and $(2k)v = kv$, this sum is equal to $(k-1)(v+w)+k(v+w)+(k-1)(v+w) = (3k-2)(v+w) = (2k-2)(v+w)$. This gives

$$\begin{aligned} z' &= m_1(n_1n_2) + n_1m_2(n_1n_2) + (2k-2)[(n_1n_2)m_1(n_1n_2) \\ &\quad +(n_1n_2)m_2(n_1n_2)] + (n_1n_2)m_2n_1 + (n_1n_2)m_2 \\ &= z \end{aligned}$$

and so $(xy)^{2k}(yx)^{2k}(xy)^{2k} = (xy)^{2k}$ as claimed.

Furthermore, folklore results show that if the variety $\mathbf{W}$ is aperiodic (i.e. contains no non-trivial groups), then for any variety $\mathbf{V}$ the groups lying in $\mathbf{V}\Box\mathbf{W}$ are exactly those in $\mathbf{V}$. In particular, $\mathbf{SL}$ is aperiodic so $(\mathbf{DO} \cap \overline{\mathbf{H}})\Box\mathbf{SL} \subseteq \overline{\mathbf{H}}$ and by the first part of our argument $(\mathbf{DO} \cap \overline{\mathbf{H}})\Box\mathbf{SL} \subseteq \mathbf{DO} \cap \overline{\mathbf{H}}$.

Let $\mathbf{V_0} = \mathbf{H}$ and $\mathbf{V_{i+1}} = \mathbf{V_i}\Box\mathbf{SL}$: to complete our proof we now show that $\mathbf{DO} \cap \overline{\mathbf{H}}$ is contained in any variety containing $\mathbf{H}$ and closed under block product with $\mathbf{SL}$. We do so by proving that any monoid of $\mathbf{DO} \cap \overline{\mathbf{H}}$ lies in some $\mathbf{V_i}$. We use the family $\sim^{\mathbf{H}}$ of congruences for $\mathbf{DO} \cap \overline{\mathbf{H}}$ and show that for each alphabet Σ with $|\Sigma| = n$, any $k \geq 0$, any group G and any word $w \in \Sigma^*$, the language $L^{(w)}_{n,k} = \{v \in \Sigma^* : v \sim^G_{n,k} w\}$ is recognized by some M in $\mathbf{V_{n+k}}$. We argue by induction on $n + k$: if $n = 0$ the claim is trivial. If $k = 0$ then L_w is the set of words that are G-equivalent to w and this can be recognized by a direct product of t copies of G where t is the number of homomorphisms from Σ^* to G.

Suppose $n, k > 0$: to check if $v \sim^G_{n,k} w$, we need to first verify that $\alpha(v) = \alpha(w)$ and, as stated in Lemma 1, this can be done using some monoid in $\mathbf{SL}$. Now, let $w = w_l a w_r$ and $v = v_l a v_r$ be the a-left-decompositions of w and v.

We claim that there is an **SL**-transduction τ and a language $K' \in L(\mathbf{V_{n+k-1}})$ such that $\tau^{-1}(K')$ is the set of words v such that $v_l \sim^G_{n-1,k} w_l$. Consider the two-element monoid $U_1 = \{0,1\}$ in **SL** with multiplication given by $0 \cdot x = x \cdot 0 = 0$ and $1 \cdot x = x \cdot 1 = x$. Let $h_l = h_r : \Sigma^* \to U_1$ be the homomorphism mapping a to 0 and every other letter to the identity 1. Thus, $h_l(v) = 0$ iff $a \in \alpha(v)$ and the transduction τ defined by h_l and h_r maps v to the sequence of triples

$$(1, v_1, 0) \dots (1, v_i, 0)(0, v_{i+1}, 0) \dots (0, v_j, 0)(0, v_{j+1}, 1) \dots (0, v_n, 1)$$

where v_i and v_j are the first and last occurrence of a in v (if any such occurrence exists). We are trying to check if $v_l \sim^G_{n-1,k} w_l$ and we know by induction that the language $L^{(w_l)}_{n-1,k}$ is in $L(\mathbf{V_{n+k-1}})$, i.e. there exists $M \in \mathbf{V_{n+k-1}}$, a homomorphism $h : \Sigma^* \to M$ and a subset $T \subseteq M$ with $L^{(w_l)}_{n,k-1} = h^{-1}(T)$. Let $h' : (U_1 \times \Sigma \times U_1)^* \to M$ be the homomorphism which maps triples $(t, b, t') \in (U_1 \times \Sigma \times U_1)$ to $h'(t, b, t') = \begin{cases} h(b) & \text{if } t = 1 \text{ and } b \neq a; \\ 1_M & \text{otherwise.} \end{cases}$

One can verify that for any $v \in \Sigma^*$ with the a-left decomposition $v_l a v_r$ we now get $h'(\tau(v)) = h'(\tau(v_l)) = h(v_l)$ because all the triples of $\tau(v)$ beyond that prefix are mapped to 1_M. Let $K' = h'^{-1}(T)$: we have $K' \in L(\mathbf{V_{n+k-1}})$ and $\tau^{-1}(K') = \{v \in \Sigma^* : v_l \in L^{(w_l)}_{n-1,k}\}$ as we had required.

Similarly, we can recognize words such that $w_r \sim^G_{n,k-1} v_r$ with the help of a U_1-transduction. We argue symmetrically about the right-decompositions and conclude that $L^{(w)}_{n,k} \in L(\mathbf{V_{n+k}})$.

4 An Application to Two-Variable Sentences

We refer the reader to e.g. [9] for a thorough overview of logical descriptions of regular languages and their relation to algebraic automata theory and circuit complexity. As we suggested in our introduction, we view finite words over the finite alphabet Σ as logical structures. We construct logical formulas using the order predicate $<$, the "content" predicates $\{Q_a | a \in \Sigma\}$, constants **t** (true) and **f** (false), Boolean connectives, a set of variables $\{x_1, \dots, x_n\}$, existential and universal quantifiers as well as modular quantifiers $\exists^{i \bmod m} x$. The atomic formulas are either **t**, **f**, $x_i < x_j$ or $Q_a x_i$. A *word structure* over alphabet Σ and variable set $\mathcal{V} \subseteq \{x_1, \dots, x_n\}$ is a pair $(w, \boldsymbol{p})$ consisting of a word $w \in \Sigma^*$ and a set of pointers $\boldsymbol{p} = (p_{i_1}, \dots, p_{i_k})$ with $1 \leq p_i \leq |w|$ which associate each variable $x_{i_j} \in \mathcal{V}$ with a position p_{i_j} in the string. A *simple extension* of a word structure $(w, \boldsymbol{p})$ over $\Sigma, \mathcal{V}$ is a word structure $(w, \boldsymbol{p'})$ over $\Sigma, (\mathcal{V} \cup \{x_{i_k+1}\})$ such that $x_{i_{k+1}} \notin \mathcal{V}$ and $p_{i_j} = p'_{i_j}$ for $1 \leq j \leq k$. We can now formally define the semantics of our formulas in a natural way. If $w = w_1 \dots w_t$ is a word, we have

$(w, \boldsymbol{p}) \models Q_a x_i$ if $w_{p_i} = a$;
$(w, \boldsymbol{p}) \models x_i < x_j$ if $p_i < p_j$;
$(w, \boldsymbol{p}) \models \exists x(\phi(x_k))$ if there exists an extension $(w, \boldsymbol{p'})$ of $(w, \boldsymbol{p})$ such that $(w, \boldsymbol{p'}) \models \phi(x_k)$;
$(w, \boldsymbol{p}) \models \exists^{i \bmod m} x_k(\phi(x_k))$ if there exists i modulo m extensions $(w, \boldsymbol{p'})$ of $(w, \boldsymbol{p})$ such that $(w, \boldsymbol{p'}) \models \phi(x_k)$;

We omit for space the obvious definition of the semantics of the Boolean connectives and of the universal quantifiers.

If ϕ is a sentence, i.e. a formula with no free variable, we denote as $L_\phi \subseteq \Sigma^*$ the language $L_\phi = \{w : (w, \emptyset) \models \phi\}$. Similarly, it will be useful for our purposes to consider the special case of formulas with a single free variable. Such a formula naturally defines a set of word structures (w, p) with $1 \leq p \leq |w|$, i.e. a *pointed language*. For any formula ϕ having a single free variable and Φ a class of such formulas, we will denote as P_ϕ the pointed language $P_\phi = \{(w, p) : (w, p) \models \phi\}$ and $P(\Phi)$ the class of all P_ϕ with $\phi \in \Phi$.

We will denote as $\mathbf{FO}$, $\mathbf{MOD}$, $\mathbf{MOD^p}$, $\mathbf{FO} + \mathbf{MOD}$ and $\mathbf{FO} + \mathbf{MOD^p}$ the class of respectively first-order sentences (no modular quantifier), modular sentences (no existential or universal quantifier), modular sentences with only mod p quantifiers and so on. The expressive power of such sentences has been investigated thoroughly and all existing results are algebraic in nature: languages definable by sentences of the fragment Γ are exactly those in $L(\mathbf{V_\Gamma})$ for some appropriate variety $\mathbf{V_\Gamma}$. (see [10] for elements of a meta-explanation). In particular, we have $L(\mathbf{FO}) = L(\mathbf{A})$ [6, 8], $L(\mathbf{MOD}) = L(\mathbf{G_{sol}})$, $L(\mathbf{MOD^p}) = L(\mathbf{G_p})$, $L(\mathbf{FO} + \mathbf{MOD}) = L(\overline{\mathbf{G_{sol}}})$ [13].

While it is natural to construct logical sentences using a new variable for each quantifier, one can just as well write sentences that reuse variables. For instance, we gave earlier a three-variable sentence defining the language $\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*$. It is also definable by the two-variable sentence $\exists x(Q_a x \wedge \exists y(Q_b y \wedge x < y \wedge \exists x(Q_c x \wedge y < x)))$. The semantics of the sentence are perfectly unambiguous (see [12] for a formal discussion). We denote $\mathbf{FO_k}$, $\mathbf{FO} + \mathbf{MOD_k}$ and so on the class of sentences using only k variables. Surprisingly, three variables suffice to describe any language in $L(\mathbf{FO} + \mathbf{MOD})$ [4, 3, 12]. If only one variable is allowed, it is easy to show that our expressive power is dramatically reduced and, for instance, that $L(\mathbf{FO_1}) = L(\mathbf{SL})$ and $L(\mathbf{MOD_1}) = L(\mathbf{Ab})$. The expressiveness of $\mathbf{FO_2}$ or $\mathbf{FO} + \mathbf{MOD_2}$ is trickier to understand [16, 12] but also admits algebraic characterizations. In order to study $\mathbf{FO} + \mathbf{MOD_2}$, [12] show that every such sentence can be rewritten so that no existential or universal quantifier appears in the scope of a modular quantifier. We will show that this is not just an artefact of the proof: at the other end of the spectrum, two-variable sentences in which no modular quantifier appears in the scope of an ordinary quantifier are provably less expressive than general $\mathbf{FO} + \mathbf{MOD_2}$ sentences.

Definition 1. *Let Σ be an alphabet and $\Phi = \{\phi_1, \ldots \phi_k\}$ be a set formulas over Σ with one free variable, say x, and a single bound variable y (possibly bound*

by more than one quantifier). A recycling *Φ-substitution σ over Σ is a function mapping two-variable sentences over the alphabet 2^Φ (the power set of Φ) to two-variable sentences over Σ as follows: each occurrence of the predicate $Q_S x$ with $S \subseteq \Phi$ is replaced by the conjunction $\bigwedge_{\phi_i \in S} \phi_i$ and each occurrence of $Q_S y$ is replaced by the similar formula in which the roles of x and y are interchanged.*

Note that recycling substitutions preserve the two-variable property. If Γ is a class of two-variable sentences and Λ is a class of formulas with one bound and at most one free variable we will denote by $\Gamma \circ \Lambda$ the class of sentences which are Boolean combinations of *sentences* in Λ and of sentences obtained by applying to a sentence ψ of Γ a recycling substitution in which all formulas in Φ lie in Λ.

Lemma 2. *Let σ be a substitution and for any $w = w_1 \dots w_n$ in Σ^* let $\sigma^{-1}(w)$ be the word $u_1 \dots u_n$ over the alphabet 2^Φ with $u_i = \{\phi_j : (w, i) \models \phi_j\}$. Then $w \models \sigma(\psi)$ iff $\sigma^{-1}(w) \models \psi$.*

The proof is omitted but is straightforward. The function mapping w to $\sigma^{-1}(w)$ is of course quite similar to the notion of transduction which we introduced in Section 2 and we now prove an equivalent of the "block product/substitution principle" of [17] which formalizes the idea:

Lemma 3. *Let Γ be a class of $\mathbf{FO} + \mathbf{MOD_2}$ sentences and Λ a class of $\mathbf{FO} + \mathbf{MOD_2}$ formulas with one free variable. If $\mathbf{V}, \mathbf{W}$ are monoid varieties such that $L(\Gamma) = L(\mathbf{V})$ and $P(\Lambda) = P(\mathbf{W})$, then $L(\Gamma \circ \Lambda) = L(\mathbf{V} \Box \mathbf{W})$.*

Proof. Since $L(\mathbf{V} \Box \mathbf{W})$ is closed under Boolean combinations, the left-to-right containment follows if we show that for any $\psi \in \Gamma$ and any Λ-substitution σ we have $L(\sigma(\psi)) \in L(\mathbf{V} \Box \mathbf{W})$. Let w be some word in Σ^* and $\Phi = \{\phi_1, \dots \phi_k\}$ be the formulas used by σ. Since $P(\mathbf{W}) = P(\Lambda)$, the pointed languages $P_{\phi_j} : \{(w, i) : (w, i) \models \phi_j\}$ can be recognized by monoids $M_1, \dots, M_k$ in $\mathbf{W}$ and $M = M_1 \times \dots \times M_k$ recognizes any Boolean combination of them. We can therefore construct an M-transduction τ such that $\tau(w_i)$ completely determines the set $\{\phi_j : (w, i) \models \phi_j\}$. Since we assume that $L(\psi)$ is recognized by a monoid N in $\mathbf{V}$, we get that $L(\sigma(\psi)) = \tau^{-1}(K)$ for some $K \subseteq (M \times \Sigma \times M)^*$ also recognized by N. Hence, $L(\sigma(\psi)) \in L(\mathbf{V} \Box \mathbf{W})$.

For the right-to-left containment, we need to show that any language of $L(\mathbf{V} \Box \mathbf{W})$ can be described by a sentence of $\Gamma \circ \Lambda$ and we proceed similarly. If τ is an M-transduction for some $M \in \mathbf{W}$ then for any triple $(m_1, a, m_2) \in M \times \Sigma \times M$, the pointed language $T_{(m_1,a,m_2)} = \{(w, i) : \tau(w_i) = (m_1, a, m_2)\}$ is in $P(\mathbf{W})$ and is thus definable by some formula $\phi_{(m_1,a,m_2)}$ in $P(\Lambda)$. Any language $K \subseteq (M \times \Sigma \times M)^*$ in $L(\mathbf{V})$ is definable by some sentence $\psi_K \in \Gamma$. Now the set of words such that $\tau(w) \in K$ is defined by the sentence obtained from ψ_K by a recycling substitution using the formulas[1] $\phi_{(m_1,a,m_2)}$.

Let $\mathbf{FF_1}$ be the class of **FO** formulas with one free and one quantified variable.

[1] Note that at any position i, *exactly one* of the $\phi_{(m_1,a,m_2)}$ is true and we can thus identify the domain of $\sigma^{-1}(w_i)$ with the set $M \times \Sigma \times M$.

Lemma 4. $P(\mathbf{FF_1}) = P(\mathbf{SL})$.

Proof (sketch). Each $\mathbf{FF_1}$ formula can be rewritten as a Boolean combination of formulas of the form $\exists y((x * y) \wedge Q_a y$ with $* \in \{<, >, =\}$. By Lemma 1, the pointed language defined by such a formula is recognized by some $M \in \mathbf{SL}$ since we are simply asking whether the letter a occurs somewhere before x (if $*$ is $>$), at x (if $*$ is $=$) or after x (if $*$ is $<$). Conversely, if a pointed language K is recognized by some $M \in \mathbf{SL}$ then by Lemma 1 membership of (w, p) in K depends on the letter w_p and on the set of letters occurring before and after the pointer. Thus, K can be defined by an $\mathbf{FF_1}$ formula.

Theorem 4. *Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is*

(1) definable by a $\mathbf{FO}+\mathbf{MOD_2}$ sentence in which no modular quantifier appears in the scope of an ordinary quantifier iff $M(L) \in \mathbf{DO} \cap \overline{\mathbf{G_{sol}}}$;
(2) definable by a $\mathbf{FO}+\mathbf{MOD_2}$ sentence in which no modular quantifier appears in the scope of another quantifier iff $M(L) \in \mathbf{DO} \cap \overline{\mathbf{Ab}}$;
(3) definable by a $\mathbf{FO} + \mathbf{MOD_2}$ sentence in which all modular quantifiers have prime moduli and no Mod_p-quantifier appears in the scope of an ordinary quantifier or of a Mod_q quantifier for p, q distinct primes iff $M(L) \in \mathbf{DO} \cap \mathbf{G_{nil}}$;

Proof. The proofs are applications of Lemma 3. By [11], we have $L(\mathbf{MOD_2}) = L(\mathbf{G_{sol}})$. Let $\mathbf{D_k}$ be the class $\mathbf{FO}+\mathbf{MOD_2}$ sentences in which no modular quantifier appears in the scope of an ordinary quantifier and in which the nesting depth of ordinary quantifiers is k. To complete our argument, it suffices to note that sentences in $\mathbf{D_k}$ are exactly those which can be obtained from $\mathbf{MOD_2}$ sentences by applying successively k $\mathbf{FF_1}$ substitutions. By applying the block-product substitution principle and Lemma 4 we get that $L(\mathbf{D_k}) = L((\ldots(\mathbf{G_{sol}}\Box\mathbf{SL})\ldots\Box\mathbf{SL}))$ (where the product has length k) and by Theorem 3 the union of the varieties on the right is exactly $\mathbf{DO} \cap \overline{\mathbf{G_{sol}}}$.

To obtain, the second and third parts of our theorem we simply need to recall that $L(\mathbf{MOD_1}) = L(\mathbf{Ab})$ and that $L(\mathbf{G_{nil}})$ is the class of languages definable by $\mathbf{MOD_2}$ sentences satisfying the restriction given in (3) [11, 13].

The regular language $K = (b*ab*a)^*b\Sigma^*$ is defined by the sentence $\exists x(Q_b x \wedge \exists^{0 \bmod 2} y(x < y \wedge Q_a x))$. We claim that K cannot be recognized by some M in $\mathbf{DO}$ and, thus defined by a $\mathbf{FO} + \mathbf{MOD_2}$ sentence in which the modular quantifiers lie outside the scope of the ordinary quantifiers. Indeed, let h be the recognizing morphism and consider $x = h(ab)$ and $y = h(a)$: we have for some ω that $h[(aba)^\omega(baa)^\omega(aba)^\omega] = h[(aba)^\omega]$ in M but this is impossible since $(aba)^\omega(baa)^\omega(aba)^\omega$ is in K while $(aba)^\omega$ is not.

The variety $\mathbf{DO}$ is decidable since it is defined by a simple identity and it is also easy to decide whether the subgroups of such a monoid lie in $\mathbf{Ab}, \mathbf{G_p}, \mathbf{G_{nil}}$ or $\mathbf{G_{sol}}$. Moreover the proof of Theorem 4 actually shows a tight correspondence between the depth of ordinary quantifiers in the $\mathbf{FO} + \mathbf{MOD_2}$ formulas with the depth of the product $(\ldots(\mathbf{H}\Box\mathbf{SL})\Box\mathbf{SL})\ldots\Box\mathbf{SL}$ and each of these varieties is also decidable, as long as $\mathbf{H}$ is. The argument requires machinery beyond the scope of this paper but the key ideas can be found in [17].

5 A New Perspective on Communication Complexity and Circuit Complexity Upper Bounds

Our decomposition of $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ in terms of block products and the ensuing logical characterization of the corresponding languages allow us to shed new light on two complexity problems recently resolved in [5, 15].

In [5], one considers Boolean circuits of bounded depth. A circuit C_n is an acyclic digraph: the nodes of in-degree 0 are *input* nodes labeled with a Boolean variable x_i with $1 \leq i \leq n$ or a Boolean constant, the other nodes are labeled by one of the Boolean functions $\{\wedge, \vee, Mod_m\}$. We also assume that there is a single *output* node, i.e. a node of out-degree 0. A circuit C_n with n inputs computes a function $C_n : \{0,1\}^n \to \{0,1\}$ in the obvious way (note that the output of a Mod_m gate is 1 if the sum of its inputs is 0 modulo m). The *depth* of C_n is the longest path from an input to the output node. The wire-size and the gate-size of C_n are, respectively, the number of edges and nodes in the graph. By extension a family of circuits $C = (C_n)_{n \geq 0}$ computes a function $C : \{0,1\}^* \to \{0,1\}$ (i.e. defines a language of $\{0,1\}^*$) and we then think of the depth and size of C as a function of n. The class of languages which can be recognized by such circuits in polynomial-size and bounded-depth is denoted ACC^0.

To process a non-Boolean input $w \in \Sigma^*$ with such circuits we can for instance require that the circuit be given a simple Boolean encoding of each letter w_i. In [5], it was shown that the regular languages definable by an ACC^0 circuit using only a linear number of wires are exactly those recognized by monoids in $\mathbf{DO} \cap \overline{\mathbf{Ab}}$. The result relies on a communication complexity result: suppose Alice and Bob are given a word $w = w_1 w_2 \dots w_{2n}$ where Alice knows only the odd-indexed w_i and Bob only the even-indexed ones. What is then the minimal number of bits that Alice and Bob need to exchange in the worst case to determine whether w lies in some regular language L? If L is recognized by some M in $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ this can be done with $O(\log n)$ bits of communication but requires $\Theta(n)$ bits otherwise [15]. It is interesting to note that the circuit complexity and communication complexity upper bounds follow simply from the block-product decomposition of this variety:

Theorem 5. *If every $K \in L(\mathbf{V})$ has a circuit with a linear number of wires then so does any $K' \in L(\mathbf{V} \Box \mathbf{SL})$. Similarly, if each $K \in L(\mathbf{V})$ has communication complexity $O(\log n)$ then so does any $K' \in L(\mathbf{V} \Box \mathbf{SL})$.*

Proof (sketch). Bilardi and Preparata show in [2] that there exists a linear-size circuit computing Prefix-Suffix-OR i.e. a circuit with n Boolean inputs $x_1, \dots, x_n$ and $2n$ outputs $p_1, \dots, p_n, s_1, \dots, s_n$ with $p_i = \bigvee_{j=1}^{i-1} x_j$ and $s_i = \bigvee_{j=i+1}^{n} x_j$.

It suffices to show that for any language K recognized by a monoid of $\mathbf{V}$ and any M-transduction τ with $M \in \mathbf{SL}$, the language $\tau^{-1}(K)$ has a linear size circuit. By Lemma 1, the value of $\tau(w_i)$ is determined entirely by the letter w_i and the sets $\alpha(w_1 \dots w_{i-1})$ and $\alpha(w_{i+1} \dots w_n)$ of letters occurring in the prefix and suffix around w_i. By using $|\Sigma|$ copies of the Prefix-Suffix-OR circuit, we can therefore build a circuit with a linear number of wires which on input $w_1 \dots w_n$

produces n blocks of $k = \log(|M|^2 \cdot |\Sigma|)$ outputs such that the ith block encodes the value of $\tau(w_i)$. These values can now simply be fed into a circuit recognizing K which, by assumption, uses only a linear number of wires.

For the communication complexity problem, Alice and Bob begin by exchanging the location of the first and last occurrence of a letter that they have access to. This requires $O(\log n)$ bits of communication and this information is enough for each player to privately compute $\tau(w_i)$ for any w_i he has access to.

In fact, the result in [2] is more general: for a regular language L, there exists a circuit which computes Prefix-Suffix-L iff L is piecewise-testable, i.e. recognized by a monoid in the variety $\mathbf{J}$. This well known variety contains $\mathbf{SL}$ and is in fact the largest variety of aperiodics satisfying the equality $\mathbf{DO}\Box\mathbf{V} = \mathbf{DO}$ [1].

Corollary 1. *Every language in* $L(\mathbf{DO} \cap \overline{\mathbf{Ab}})$ *can be defined by a linear-wire-size family of* ACC^0 *circuits and has two-party communication complexity* $O(\log n)$.

Proof. For any $K \in L(\mathbf{Ab})$, membership of $w \in K$ depends solely on the number $|w|_a$ of occurrences of each letter a in w modulo some integer m [7]. Computing $|w|_a$ modulo m, can clearly be done with a single Mod_m gates with n input wires and by a communication protocol of cost $2 \log |m|$. Thus, any K in $L(\mathbf{Ab})$ has linear-wire-size circuits and communication complexity $O(1)$. Our statement then follows from Theorem 5 and the decomposition result of Theorem 3.

We should note that the same idea can be used to show that for every language L recognized by $M \in \mathbf{DO} \cap \overline{\mathbf{G_{nil}}}$ there exists k such that the k-party communication complexity of L is $O(1)$ (see [14] for a discussion of the multiparty communication model).

Alternatively, one can view the linear-wire-size circuits for $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ as evaluating a two-variable formula with no modular quantifier nested in another quantifier. First the circuit evaluates the most deeply nested subformulas with one free variable, say x. These are $\mathbf{FF_1}$ formulas and thus Boolean combinations of formulas such as $\phi(x) = \exists y(x < y \wedge Q_a y)$ and the Prefix-Suffix-OR construction allows us to simultaneously compute the value of $\phi(x)$ for *each* value of x using only $O(n)$ wires. These results are used in the next step to compute, for *all* positions y the value of formulas such as $\psi(y) = \exists x(x < y \wedge \phi(x))$. At the output level, we evaluate the value of a modular quantifier $\exists^{i \bmod m} x \chi(x)$. It suffices to feed the n values of $\chi(x)$ obtained in the previous step into a Mod_m gate.

In general, recognizing a language definable by an $\mathbf{FO} + \mathbf{MOD_2}$ sentence in which modular quantifiers *are* allowed in the scope of other quantifiers will require an ACC^0 circuit with a super-linear number of wires. However, these languages can still be recognized using only $O(n)$ *gates*. The strategy to build such a circuit is similar: we are given, at some stage in our circuit the values for each x of a formula $\phi(x)$ and want to compute for each y the value of a formula of the form $\psi(y) = \exists^{i \bmod m} x(x < y \wedge \phi(x))$. For each y, this can of course be computed by a *single* Mod_m gate in which we feed the results of $\phi(x)$ for each $x < y$. The sub-circuit computing all values $\psi(y)$ from the values $\phi(x)$ will use only n gates but $\Omega(n^2)$ wires. In fact, we conjecture that the

regular languages recognized by an ACC^0 circuit with a linear number of gates are exactly those definable in $\mathbf{FO} + \mathbf{MOD_2}$ and that, among these, the fine line between linear-gate-size and linear-wire-size is thus captured exactly by the ability to pull modular quantifiers out of the scope of any other quantifier.

It is also very natural to try and characterize regular languages which can be computed by linear-gate-size restricted ACC^0 circuits where we allow only Mod_m gates (i.e. CC^0 circuits), or only $\wedge, \vee$ gates (i.e. AC^0 circuits). We believe that these regular languages are exactly those in $\mathbf{MOD_2}$ and $\mathbf{FO_2}$ respectively. In particular, we conjecture that every regular language recognized by an AC^0 circuit with a linear number of gates can also be recognized by an AC^0 circuit with a linear number of wires. Resolving this question for CC^0 circuits amounts to proving a lower bound of $\omega(n)$ for the number of gates in CC^0 circuits computing the AND of n bits or the word problem of a non-solvable group. To the best of our knowledge, the state-of-the-art lower bound state that CC^0 circuits for AND requires $\Omega(n)$ non-input gates, a world away from the suspected $\Omega(c^n)$. As a first step, it would an interesting start to establish an $\omega(n)$ lower bound for the number of wires in such circuits.

References

1. J. Almeida and P. Weil. Profinite categories and semidirect products. *J. Pure and Applied Algebra*, 123:1–50, 1998.
2. G. Bilardi and F. Preparata. Characterization of associative operations with prefix circuits of constant depth and linear size. *SIAM J. Comput.*, 19(2):246–255, 1990.
3. N. Immerman and D. Kozen. Definability with a bounded number of bound variables. *Information and Computation*, 83(2):121–13, 1989.
4. J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Berkeley, 1968.
5. M. Koucký, P. Pudlák, and D. Thérien. Bounded-depth circuits: separating wires from gates. In *Symposium on Theory of Computing (STOC'05)*, 2005.
6. R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, Cambridge, Mass., 1971.
7. J.-E. Pin. Syntactic semigroups. In G. R. et A. Salomaa, editor, *Handbook of language theory*, volume 1, chapter 10, pages 679–746. Springer Verlag, 1997.
8. M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, Apr. 1965.
9. H. Straubing. *Finite Automata, Formal Logic and Circuit Complexity*. Boston: Birkhauser, 1994.
10. H. Straubing. On the logical description of regular languages. In *Proc. of the 5th Latin American Theoretical Informatics Conf. (LATIN '02)*, 2002.
11. H. Straubing and D. Thérien. Weakly iterated block products of finite monoids. In *Proc. of the 5th Latin American Theoretical Informatics Conf. (LATIN '02)*, 2002.
12. H. Straubing and D. Thérien. Regular languages defined by generalized first-order formulas with a bounded number of bound variables. *Theory of Computing Systems*, 36(1):29–69, 2003.
13. H. Straubing, D. Thérien, and W. Thomas. Regular languages defined by generalized quantifiers. *Information and Computation*, 118:289–301, 1995.

14. P. Tesson. *Computational Complexity Questions Related to Finite Monoids and Semigroups*. PhD thesis, McGill University, 2003.
15. P. Tesson and D. Thérien. Complete classifications for the communication complexity of regular languages. *Theory of Comput. Syst.*, 2004. To appear.
16. D. Thérien and T. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 256–263, 1998.
17. D. Thérien and T. Wilke. Nesting until and since in linear temporal logic. *Theory Comput. Syst.*, 37(1):111–131, 2004.