# Modular Quantifiers

Howard Straubing[1]
Denis Thérien[2] *

[1] Computer Science Department
Boston College
Chestnut Hill, Massachusetts 02476
USA

[2] School of Computer Science
McGill University
Montréal,Québec
CANADA

straubin@cs.bc.edu,denis@cs.mcgill.ca

In the late nineteen-eighties much of our research concerned the application of semigroup-theoretic methods to automata and regular languages, and the connection between computational complexity and this algebraic theory of automata. It was during this period that we became aware of the work of Wolfgang Thomas. Thomas had undertaken the study of concatenation hierarchies of star-free regular languages—a subject close to our hearts— by model-theoretic methods. He showed that the levels of the dot-depth hierarchy corresponded precisely to level of the quantifier alternation hierarchy within first-order logic[26], and applied Ehrenfeucht-Fraïssé games to prove that the dot-depth hierarchy was strict [27], a result previously obtained by semigroup-theoretic means [4, 18].

Finite model theory, a subject with which we'd had little prior acquaintance, suddenly appeared as a novel way to think about problems that we had been studying for many years. We were privileged to have been introduced to this field by so distinguished a practitioner as Wolfgang Thomas, and to have then had the opportunity to work together with him. The study of languages defined with modular quantifiers, the subject of the present survey, began with this collaboration.

## 1 Generalized First-order Formulas over $<$

### 1.1 First-Order Logic and Star-free Sets

A little background first, about the ordinary kind of quantifier: Properties of words over a finite alphabet $\Sigma$ can be expressed in predicate logic by interpreting variables as positions $\{0, 1, \ldots, n-1\}$ in a string of length $n$,

* The second author was supported by grants from NSERC and FQRNT

and including for each $\sigma \in \Sigma$ a unary relation symbol $Q_\sigma$, where $Q_\sigma x$ is interpreted to mean that the letter in position $x$ is $\sigma$. The signature typically includes other predicate symbols—“numerical predicates”—that are independent of the letters that appear in the positions concerned, but instead allow us to talk about relations between positions. It is quite interesting to see what happens when we modify this part of the signature, but for now we will suppose that there is just one such predicate symbol: $<$, denoting the usual order on the positions.

A sentence of first-order logic thus defines a language in $\Sigma^*$, namely the set of all strings that satisfy the sentence. For example, if $\Sigma = \{\sigma, \tau\}$, then the sentence

$$\exists x(Q_\sigma x \wedge \neg\exists y(y < x))$$

says that there is a position containing the letter $x$ preceded by no other position, and thus defines the language $\sigma\Sigma^*$ of words whose first letter is $\sigma$.

Likewise the sentence

$$\begin{aligned} \exists x(Q_\sigma x \wedge \neg\exists y(y < x)) \quad & \wedge \quad \exists x(Q_\tau x \wedge \neg\exists y(y > x)) \\ & \wedge \quad \forall x\Big(Q_\tau x \leftrightarrow \exists y\big(Q_\sigma y \wedge (y < x) \\ & \wedge \quad \forall z(z > y \rightarrow \neg(z < x))\big)\Big) \end{aligned}$$

says that the first letter is $\sigma$, the last letter is $\tau$, and that the positions containing $\tau$ are those immediately following positions containing $\sigma$. This defines the language $(\sigma\tau)^*$.

Now $(\sigma\tau)^*$ is a *star-free* subset of $\{\sigma, \tau\}^*$. This means that it can be defined by an extended regular expression in which arbitrary boolean operations are permitted along with concatenation, but in which the star operation is not used. This may not be obvious at first, since we have certainly used the star to write it! But in fact this language is identical to

$$\big(\tau\varnothing^c \cup \varnothing^c\sigma \cup \varnothing^c(\sigma\sigma \cup \tau\tau)\varnothing^c\big)^c,$$

where the superscript $c$ denotes complementation in $\{\sigma, \tau\}^*$.

McNaughton and Papert [13] showed that the star-free languages are exactly those definable by first-order sentences over $<$ . It is not hard to see how to express the concatenation operation in first-order logic, so let us concentrate instead on why the converse is true. Our account here is inspired by the treatment in Thomas [26]. If $w_1, w_2 \in \Sigma^*$, and $k \geq 0$, then we write $w_1 \equiv_k w_2$ to mean that $w_1$ and $w_2$ satisfy all the same sentences of quantifier depth $k$. We write $[w]_k$ to denote the equivalence class of the word $w$ under this relation. One can now show that for any word $v$,

$$[v]_{k+1} = \bigcap_{*}[v_1]_k\sigma[v_2]_k - \bigcup_{**}[u_1]_k\sigma[u_2]_k.$$

Here, the index set $*$ in the intersection is the set of all triples $(v_1, \sigma, v_2)$ such that $\sigma \in \Sigma$ and $v = v_1\sigma v_2$, and the union over the set of all triples $(u_1, \sigma, u_2)$ such that $v$ does not have a factorization $v_1\sigma v_2$ with $v_i \equiv_k u_i$ for $i = 1, 2$. This can be established with a by-now routine argument using games. Since $\equiv_k$ has finite index, the intersection is in fact a finite intersection. So the above equation shows that for all $k$, each $\equiv_k$-class is a star-free language. Since a language defined by a sentence of depth $k$ is a finite union of such classes, we get the desired result.

Observe that the argument outlined above makes no use of the *other* characterization of star-free languages, namely Schützenberger's Theorem that these are exactly the languages whose syntactic monoids are aperiodic (*i.e.,* contain no nontrivial groups) [16]. But algebra and semigroups are not completely absent, for the equivalences $\equiv_k$ are congruences of finite index on $\Sigma^*$, and the content of Schützenberger's Theorem shows in essence that the quotient monoids of these congruences generate all the finite aperiodic monoids.

## 1.2 Counting Factorizations

Earlier (in our Ph.D. dissertations!) we had both studied a variant of the concatenation operation that counted factorizations modulo some period: Let $L_1, L_2 \subseteq \Sigma^*$, $\sigma \in \Sigma$, and $0 \leq r < 1$. Then we define $(L_1, \sigma, L_2, r, q)$ to be the set of words $w$ for which the number of factorizations $w = w_1\sigma w_2$ if congruent to $r$ modulo $q$.

In our discussions with Thomas we realized that the precise power of this operation could be captured if one introduced *modular quantifiers* into our logical languages:

$$\exists^{r \bmod q} x\varphi(x)$$

is interpreted to mean 'the number of positions $x$ for which $\varphi(x)$ holds' is congruent to $r$ modulo $q$.

As an example, consider the sentence

$$\exists^{0 \bmod 3} x(Q_\tau x \wedge \exists^{0 \bmod 2} y(Q_\sigma y \wedge y < x)).$$

This defines the set of all strings in which the number of occurrences of $\tau$ preceded by an even number of $\sigma$ is divisible by 3. Observe that this particular sentence uses modular quantifers exclusively, and that it is possible to rewrite it so that it only uses modular quantifiers of modulus 6.

We were able to adapt the argument given above for star-free languages to this new quantifer: Let us fix a modulus $q$, and let us redefine $v_1 \equiv_k v_2$ to mean that $v_1$ and $v_2$ satisfy the same sentences of quantifier depth $k$, where we now allow modular quantifiers of modulus $q$ as well as ordinary quantifiers. Let $L$ be defined by the sentence

$$\exists^{r \bmod q} x\varphi(x),$$

where $\varphi$ has depth $k$. We showed that $L$ is a boolean combination of languages of the form $(K, \sigma, K', s, q)$, where $K$ and $K'$ are $\equiv_k$-classes. The same conclusion holds if we define $\equiv_k$ in terms of modular quantifiers exclusively.

It readily follows that languages constructed using boolean operations and ordinary concatenation together with the operations $(L_1, \sigma, L_2, r, q)$ are exactly those defined by sentences using both ordinary and modular quantifiers, and that languages defined using the operations $(L_1, \sigma, L_2, r, q)$ alone are exactly those definable using only modular quantifiers.

Our real interest, however, stemmed from the fact that these language classes could all be characterized effectively in semigroup-theoretic terms. The example language defined above with quantifiers of modulus 2 and 3 was derived from a descripiton of the set of words in the permutations $\sigma = (1, 2)$ and $\tau = (1, 2, 3)$ that evaluate to the identity in the symmetric group $S_3$. This works in general for finite solvable groups, for we can derive such descriptions of word problems from the composition series for the groups. It turns out that the languages definable using only modular quantifiers are exactly the languages whose syntactic monoids are solvable groups, and those definable using both modular and ordinary quantifiers are exactly those whose syntactic monoids contain only solvable groups.

Let us denote by $FO[<]$ the family of languages definable by first-order sentences over $<$, by $(FO + MOD_q)[<]$ those definable with both ordinary first-order quantifiers and modular quantifiers of modulus $q$, and by $MOD_q[<]$ those definable using only modular quantifiers of modulus $q$. (We assume all of this is with respect to a fixed finite alphabet $\Sigma$.)

**Theorem 1.1.** (Straubing, Thérien and Thomas [22]) $MOD_q[<]$ is the family of regular languages whose syntactic monoids are solvable groups of cardinality dividing a power of $q$. $(FO + MOD_q)[<]$ is the family of regular languages $L$ such that every group in $M(L)$ is a solvable group of cardinality dividing a power of $q$.

This second of these facts is far deeper than the first: While a solvable group by definition decomposes into a sequence of extensions by cyclic groups, which generates the expression in terms of modular quantifiers, the existence of a comparable decomposition for monoids that contain solvable groups requires the use of the Krohn-Rhodes Theorem [12].

The result of this is that we are able to effectively decide if a regular language, given, let us say, by a regular expression or an automaton, can be defined by a sentence involving modular quantifiers, and if so actually produce the sentence. For instance, suppose $L$ is recognized by a deterministic automaton with four states. We can explicitly write down a Krohn-Rhodes decomposition of the monoid of all transformations on a four-element set into factors that are either small aperiodic monoids or cyclic groups of order two or three. This can be used to produce a sentence for $L$ containing ordinary quantifiers along with modular quantifiers of modulus 2 and 3. In contrast, if the minimal DFA for $L$ has five states, and if the transition monoid contains all the even permutations of the states, then no such sentence for $L$ is possible, irrespective of the moduli used.

## 1.3 Quantifiers and the Block Product

The two-sided decomposition theory for finite monoids developed by Rhodes and Tilson [14] permits a deep understanding of the connection between logic and algebra that underlies Theorem 1.1. Suppose that $M$ and $N$ are two finite monoids. We write the operation in $M$ additively, and its identity as 0. This is not meant to imply that $M$ is commuative, although in fact, in the critical examples we consider below, $M$ will be commutative. We consider both a left action and a right action of $N$ on $M$ that are compatible in the sense that

$$(nm)n' = n(mn')$$

for all $m \in M$; $n, n' \in N$. We further suppose that these actions respect the identities in both monoids, so that

$$n0 = 0n = 0$$

for all $n \in N$, and

$$1m = m1 = m$$

for all $m \in M$. The *bilateral semidirect product* $M ** N$ with respect to these actions is the monoid whose underlying set is $M \times N$ and whose operation is given by

$$(m_1, n_1)(m_2, n_2) = (m_1 n_2 + n_1 m_2, n_1 n_2).$$

Rhodes and Tilson also define a *block product* $M \Box N$, a bilateral semidirect product of $M^{N \times N}$ and $N$ that contains all the bilateral semidirect products $M ** N$.

The connection with quantification comes in when we consider languages recognized by bilateral semidirect products $M ** N$ (or, what is the same thing, block products $M \Box N$) in which $M$ is either idempotent and commutative, or an abelian group. This becomes clear if we try to compute the image of a word $w = \sigma_1 \cdots \sigma_r$ under a homomorphism into the bilateral semidirect product. If we suppose that this morphism maps $\sigma_i$ to $(m_i, n_i)$, then $w$ is mapped to:

$$\prod_{i=1}^{r}(m_i, n_i) = \Big(\sum_{i=1}^{r}(\prod_{j=1}^{i-1} n_j)m_i(\prod_{k=i+1}^{r} n_k), \prod_{i=1^r} n_i\Big).$$

In other words, computation in $M ** N$ keeps track in $M$ of the factorizations $w = u\sigma v$, where the images of $u$ and $v$ are computed in $N$. It follows that if $M$ is idempotent and commutative, then a language recognized by $M ** N$ is a boolean combination of languages of the form $L\sigma L'$, where $L, L'$ are recognized by $N$; and that if $M$ is an abelian group of exponent $q$, then any language recognized by $M ** N$ is a boolean combination of languages of the form $(L, \sigma, L', r, q)$, where again $L$ and $L'$ are recognized by $L$. As mentioned above, these language operations can be captured by application of ordinary and modular quantifiers.

Conversely, consider a language $L$ defined by a sentence of the form

$$\exists x\varphi,$$

or

$$\exists^{r \bmod q} x\varphi,$$

where $\varphi$ is itself a formula with ordinary and modular quantifiers over the signature $\{<\}$. We can view the formula $\varphi$, which has a single free variable $x$, as defining a language $L_\varphi$ over the extended alphabet $\Sigma \times 2^{\{x\}}$. Elements of this language are words in which one of the positions is marked, and which satisfy $\varphi$ when the free variable is instantiated by the marked position. Likewise, we can view a formula with $k$ free variables as definining a language of marked words with $k$ distinct marks, some of which may coincide. Let $\mu_L : \Sigma^* \to M(L)$ be the syntactic morphism of $L$, and $\nu_L : A^* \to M(L')$.

This is where the decomposition theory of Rhodes and Tilson comes in. The relation $\nu_L \mu_L^{-1} : M(L) \to M(L')$ is a *relational morphism,* and its *kernel category* is idempotent and commutative (in the case of ordinary quantifiers) or covered by an abelian group of exponent $q$ (in the case of modular quantifiers). This implies that $M(L)$ is recognized by a block product $K \square M(L')$, where $K$ is idempotent and commutative or an abelian group of exponent $q$, depending on the quantifiers.

Theorem 1.1 follows from these observations and the Krohn-Rhodes Theorem for block products: The solvable groups of order dividing a power of $q$ for the smallest variety of finite monoids closed under block product and containing all the abelian groups of exponent $q$, and all the idempotent and commutative monoids. This is the approach taken in the journal version of our paper with Wolfgang Thomas [23], and in Straubing [19], which considers a large assortment of regular language classes defined with modular quantifiers.

## 2 Circuits

### 2.1 Constant-depth Circuits and the $ACC^0$ Problem

Why study modular quantifiers in the first place? To be frank, when we began our work we did not have a particularly compelling answer to this question! Modular counting of factorizations was an instance of an operation that happened to be easy to describe, but not particularly easy to understand, which we were able to analyze completely with our new algebraic methods.

But, as sometimes happens when you are lucky, we subsequently found a very good reason to be interested in these matters. This came from computational complexity.

A *circuit* with $n$ inputs is a directed acyclic graph, and in our circuits we will require that there be a single sink node. Each source node is labeled by a variable $x_i$ or its negation $\neg x_i$, and each non-source node of in-degree $r$ by a function $f : \{0,1\}^r \to \{0,1\}$. Initially we will just use the $r$-ary $AND$ and $OR$ functions, corresponding to standard logic gates, but later we will play around with the gate type.

The circuit computes as follows: Given a bit string $a_1 \cdots a_n$, place $a_i$ at each source node labeled $x_i$, $\neg a_i$ at each source node labeled $\neg x_i$, and recursively compute a bit value for each non-source node: If the entering edges of a node labeled $f$ connect to nodes with bit values $b_1, \ldots, b_r$, then the node will get the value $f(b_1, \ldots, b_r)$. (In all of our examples the gate functions $f$ are symmetric, so we needn't worry about ordering the incoming edges to a node.) The input is accepted if the bit value assigned to the sink node is 1, and rejected otherwise.

A circuit family with one circuit for each positive input length $n$ thus recognizes a language $L \subseteq \{0,1\}^*$. If the circuits in the family contain only $AND$ and $OR$ gates, the depth of the circuits in the family (the length of the longest path from an input to the sink) is bounded by a constant, and the size (the number of nodes) of the $n^{th}$ circuit in the family is bounded by $n^k$ for some constant $k$, then the language is said to belong to the class $AC^0$.

$AC^0$ contains the set of all strings of the form $uv$, where $|u| = |v|$ and the integer with binary representation $u$ is greater than the integer with binary representation $v$. Indeed, we can write this circuit explicitly as

$$\bigvee_{j=1}^{n} \Big( \bigwedge_{i<j} \big( (x_i \wedge x_{i+n}) \vee (\neg x_i \wedge \neg x_{i+n}) \wedge x_j \wedge \neg x_{j+n} \big) \Big).$$

If we were to allow multiple outputs, then we could use the same strategy to perform binary addition of two $n$-bit numbers in depth 3 and size $n^{O(1)}$. $AC^0$ contains every star-free regular language in $\{0,1\}^*$, and in fact every star-free regular language over any finite alphabet $\Sigma$, provided we adopt a fixed-length encoding of letters of $\Sigma$ by bit strings.

Let us contrast $AC^0$ with another circuit complexity class, this one called $NC^1$. $NC^1$ also consists of polynomial-size families of circuits with $AND$ and $OR$ gates, but we allow the depth of the circuits to grow logarithmically (*i.e.,* the depth of the $n^{th}$ circuit is $O(\log n)$) and we require every node to have in-degree 2. $NC^1$ contains every regular language. If we were to allow multiple outputs, then we could multiply two $n$-bit numbers or add $n$ $n$-bit numbers [5] and even *multiply* $n$ $n$-bit numbers and perform integer division [3].

The natural question in computational complexity is whether one model is really computationally more powerful than another. It is easy to see that $AC^0$ is contained in $NC^1$. Can we really do more with logarithmic-depth circuits?

Furst, Saxe and Sipser [6] showed that, indeed, the language $PARITY$, consisting of all bit strings with an even number of 1's, requires superpolynomial-size circuit families of constant depth, and thus is not in $AC^0$. The same argument shows that for any $q > 1$, the set of bit strings in which the number of 1's is divisible by $q$ is not in $AC^0$, and a reduction argument shows that we cannot do such things as multiply two integers in multiple-output $AC^0$.

We can try to boost the power of the constant-depth model by adding things like $PARITY$ as a kind of oracle gate. More formally, we let $q > 1$ and consider the functions $f_r : \{0,1\}^r \rightarrow \{0,1\}$ where $f_r(a_1, \ldots, a_r) = 1$ if and only if $a_1 + \cdots + a_r$ is divisible by $q$. We call such a function a $MOD_q$ *gate.* $ACC^0(q)$ is the family of languages recognized by constant-depth polynomial-size families of circuits that include $AND$, $OR$ and $MOD_q$ gates. $ACC^0$ is the union of the classes $ACC^0(q)$ over all $q > 0$.

The definitive result on $ACC^0$ is the following theorem of Smolensky [17], which contains the result of Furst-Saxe and Sipser as a special case.

**Theorem 2.1.** Let $p$ and $q$ be distinct primes, and $k > 0$. The the set $L_p$ of bit strings in which the number of 1's is divisible by $p$ is not in $ACC^0(q)$.

But that's not really definitive enough! It tells us that we cannot count, say, modulo 7 in $ACC^0(8)$, or $ACC^0(25)$, but tells us nothing about whether we can do this in $ACC^0(6)$, because 6 has two distinct prime factors. We *expect* that it cannot be done in $ACC^0(6)$, and more generally:

**Conjecture 2.2.** (a) Let $q > 1$. If $p$ is a prime that does not divide $q$, then $ACC^0(q)$ does not contain $L_p$.
(b) $ACC^0$ is properly contained in $NC^1$.

We know very little about what occurs when the modulus of the modular gates is not a prime power. Not only has this problem remained unsolved for twenty years, but it stands, in a sense, at the very frontier of current knowledge about computational complexity. We do not know how to separate $NC^1$ from $ACC^0$, but we also do not know if there is a language in $LOGSPACE$ that is not in $NC^1$, nor a language in $P$ that is not in $LOGSPACE$, nor, of course, a language in $NP$ that is not in $P$. It is entirely consistent with the current state of our knowledge that $ACC^0$ contains an $NP$-complete problem.

## 2.2 Circuits and Predicate Logic

There is a close connection between the constant-depth circuit families we described above, and formulas of first-order logic used to define languages, first observed by Gurevich and Lewis [7], and independently by Immerman [8].

We illustrate this with an example. Let us return to the language

$$L_{comp} = \{uv : |u| = |v|, (u)_2 > (v)_2\},$$

where $(w)_2$ denotes the integer whose binary representation is $u$. In the last section we gave a description of a circuit family recognizing this language.

If we are allowed to read $u$ and $v$ in parallel then we could consider the pair $(u, v)$ as a string of length $n = |u| = |v|$ over the four-letter alphabet $\{0,1\}\times\{0,1\}$. With this interpretation, $L_{comp}$ is a star-free regular language, defined by the first-order sentence

$$\exists z_1(Q_{(1,0)}z_1 \wedge \forall z_2((z_2 < z_1) \rightarrow Q_{(1,1)}z_2 \vee Q_{(0,0)}z_2)).$$

Of course, positions in this string encode pairs of positions in $uv$, and we can translate this into a sentence that talks directly about $uv$:

$$\begin{aligned} \exists x_1 \exists y_1 (Q_1 x_1 \quad & \wedge \quad Q_0 y_1 \wedge (y_1 = x_1 + n) \\ & \wedge \quad \forall x_2 \forall y_2 (x_2 < x_1 \wedge y_2 = x_2 + n \rightarrow \\ & \qquad (Q_1 x_2 \wedge Q_1 y_2) \vee (Q_0 x_2 \wedge Q_1 y_2))) \end{aligned}$$

The result is a first-order sentence that defines the original language $L_{comp}$. Observe that we have had to introduce a new numerical predicate $y = x + n$ which says that $x$ and $y$ occupy corresponding positions in the two halves of $uv$.

Conversely, we can 'unroll' this first-order formula and obtain expressions for a circuit family recognizing $L_{comp}$. These will be much like the ones that we saw in the last section.

This sort of argument works in general: if we denote by $\mathcal{N}$ the family of all numerical predicates, then $AC^0$ is exactly the same as the class $FO[\mathcal{N}]$ of languages defined by first-order sentences with no restriction on numerical predicates. The identical argument works if we permit modular quantifiers of modulus $q$ in our formulas and $MOD_q$ gates in our circuits. The details are given in Barrington, *et. al.*[2].

As a result, we have:

**Theorem 2.3.**
$$ACC^0(q) = (FO + MOD_q)[\mathcal{N}].$$

## 2.3 The Connection with Regular Languages

A consequence of the theorem of Furst, Saxe and Sipser cited above, noted in [2], is that the regular languages in $AC^0$ are precisely those definable by first-order sentences in which, in addition to the order relation, there are predicates $\equiv_t$ for equivalence of positions modulo $t$, for all positive integers $t$. In [19], the numerical predicates that are definable by first-order formulas in $<$ and $\equiv_t$ are called *regular numerical predicates,* since this is in fact the largest class of numerical predicates that one can introduce into sentences and still guarantee that every definable language is regular.We denote by $\mathcal{R}$ this class of numerical predicates.

The languages definable in this way are not quite star-free, since they include, in particular, the languages $(\{0,1\}^t)^*$ of strings of length divisible by $t$. But they are almost star-free in the sense that they are the smallest class containing the star-free languages and the languages $(\{0,1\}^t)^*$ that is closed under boolean operations and concatenation. If we combine this with the logical characterization of $AC^0$, we obtain:

**Theorem 2.4.** The family of regular languages in $FO[\mathcal{N}]$ is $FO[\mathcal{R}]$.

It is therefore reasonable to conjecture

**Conjecture 2.5.** Let $q > 0$. The family of regular languages in $(FO + MOD_q)[\mathcal{N}]$ is $(FO + MOD_q)[\mathcal{R}]$.

In fact, this is equivalent to our previous Conjecture 2.2. The principal reason for this equivalence is the fact, discovered by Barrington [1], that languages whose syntactic monoids contain a nonsolvable group are complete for $NC^1$ under a particularly restrictive kind of reduction: a consequence is that as soon as $(FO + MOD_q)[\mathcal{N}]$ contains such a regular language, it contains all of $NC^1$.

We have thus reduced our conjectured solution to one of the outstanding open problems in computational complexity to a purely model-theoretic question about the definability of regular languages in an extension of first-order logic. It makes sense to look for a model-theoretic explanation of the phenomenon. Unfortunately, the only proof we that we possess for the pure first-order case, Theorem 2.4 requires the lower bounds results from circuit complexity. And, as we have already remarked, none of the methods for proving these bounds generalizes to treat $ACC^0$.

There has been some small progress on the question. Roy and Straubing [15] use model-theoretic collapse results to prove Conjecture 2.5 when the only numerical predicate allowed is the addition of positions. They also show the conjecture holds for sentences that contain only the order relation and arbitrary monadic numerical predicates. However, as they discuss, there are fundamental obstacles to generalizing these methods.

## 3 Sentences with a Bounded Number of Variables

### 3.1 Two- and Three-Variable First-Order Sentences

An occurrence of a variable $x$ in a sentence can lie within the scope of several different quantifiers that use this variable. It is only the innermost such quantifer that binds this occurrence of $x$. Thus it is possible to re-use variables within a sentence. For instance, the sentence

$$\exists x(Q_\sigma x \wedge \exists y(y < x \wedge Q_\tau y \wedge \exists x(x < y \wedge Q_\tau x \wedge \exists y(y < x \wedge Q_\sigma y))))$$

defines the set of all strings that have a subsequence $\sigma\tau\tau\sigma$.

It is known that every first-order sentence over $<$ is equivalent to one in which only three variables are used. (Kamp [10], Immerman and Kozen [9]). Thérien and Wilke showed that the languages definable by two-variable sentences could be characterized in terms of the syntactic monoid [25]: These are the languages whose syntactic monoids belong the the variety **DA**. There are many equivalent definitions of this class of monoids, but here is one we will find most useful: Two elements $m$ and $n$ of a monoid $M$ are said to be $\mathcal{J}$-equivalent if $MmM = MnM$. A monoid is in **DA** if it is aperiodic, and if every element $\mathcal{J}$-equivalent to an idempotent is itself idempotent.

The language $(\sigma\tau)^*$ that we discussed earlier serves as a good example that separates two-variable definability from first-order definability. It is quite plausible that we cannot define this language without referring to one position being between two others, and that this will require three variables to do. The proof is that the words $\sigma$ and $\sigma\tau\sigma$ represent the same elements of the syntactic monoid of this language, and so $\sigma$ and $\sigma\tau$ are $\mathcal{J}$-equivalent, but the second of these is idempotent, while the first is not.

### 3.2 Modular Quantifiers with a Bounded Number of Variables

In [21] we investigated what happens when we bound the number of variables in sentences that contain modular quantifiers. If modular quantifiers are used exclusively, then every sentence is equivalent to one in which only two variables are used. When both modular and ordinary quantifiers are allowed, then three variables are again sufficient to define all the languages in $(FO+MOD)[<]$. An interesting phenomenon occurs in the two-variable case. Consider again the language $(\sigma\tau)^*$ in the example above. It is defined by a sentence that says the length of the string is even, and that a position contains $\tau$ if and only if it is an odd-numbered position:

$$\exists^{0 \bmod 2} x(x = x) \wedge \forall x(Q_\tau x \leftrightarrow \exists^{0 \bmod 2} y(y < x)).$$

What is remarkable here is that modular quantifiers are not required at all to define this language, but allowing them leads to a more economical (in terms of the number of variables) specification. Furthermore, appearances to the contrary, the modulus used is irrelevant. It is possible to define the same language with two variables using modular quantifiers of modulus 3, a puzzle we leave for the reader.

Let us denote by $(FO + MOD)^2[<]$ the family of languages in $(FO + MOD)[<]$ definable by a two-variable sentence. We further denote by $\Sigma_2[MOD]$ the family of languages defined by sentences over $<$ in which there is a block of existential quantifiers, followed by a block of universal quantifiers, followed by a formula in which only modular quantifiers appear. The family $\Pi_2[MOD]$ is defined similarly. We showed:

**Theorem 3.1.** Let $L \subseteq \Sigma^*$. The following are equivalent
(a) $L \in (FO + MOD)^2[<]$.
(b) $L \in \Sigma_2[MOD] \cap \Pi_2[MOD]$.
(c) The syntactic monoid $M(L)$ divides a wreath product $M \circ G$, where $M \in$ **DA** and $G$ is a solvable group. (That is, $M(L)$ belongs to the pseudovariety $\mathbf{DA} * \mathbf{G}_{sol}$.)

Interestingly, we do not know how to determine effectively if a given finite monoid belongs to $\mathbf{DA} * \mathbf{G}_{sol}$. The problem is equivalent to determining whether a set of partial one-to-one functions on a finite set $X$ can be extended to a solvable permutation group on a larger set $Y$. We refer the reader to [21] for a discussion of this problem, as well as an apparent connection to computational complexity; and also to [20], where we give a different proof of the equivalence of $(a)$ and $(c)$ above, based on the block product.

On the other hand, we do possess an effective test for whether a given finite monoid $M$ divides a wreath product of a monoid in **DA** and a finite group (which may not be solvable): If $e$ and $f$ are $\mathcal{J}$-equivalent idempotents of $M$, and $ef$ lies in the same $\mathcal{J}$-class, then $ef$ is itself idempotent. To see how this criterion works in an example, consider the language $L = (\sigma{+}\tau){*}\sigma\sigma(\sigma{+}\tau){*}$ of all strings over $\{\sigma, \tau\}$ in which there are two consecutive occurrences of $\sigma$. Since $\tau$ and $\tau\sigma\tau$ are equivalent in $M(L)$, as are $\sigma$ and $\sigma\tau\sigma$, we conclude that $\sigma$, $\tau$, $\sigma\tau$ and $\tau\sigma$ are in the same $\mathcal{J}$-class. Of these, all but $\sigma$ are idempotent. The condition is then violated by choosing $e = \sigma\tau$ and $f = \tau\sigma$, since the product $ef$ is equal to the non-idempotent $\sigma$.

We conclude that this language requires three variables to define, even if modular quantifiers are permitted. Observe how this purely model-theoretic conclusion, which might be difficult to obtain otherwise, follows from a relatively simple calculation in the minimal automaton of $L$.

## 3.3 The placement of the modular quantifiers, and more circuit complexity

An important element in the proof of Theorem 3.1 above is a kind of normal form for two-variable sentences over $<$ containing modular quantifiers: Every sentence of $(FO + MOD)^2[<]$ is equivalent to one in which an ordinary quantifier never appears within the scope of a modular quantifier.

We therefore would expect the expressive power of two-variable logic to decrease if we require instead that modular quantifiers not appear inside the scope of other quantifiers. Tesson and Thérien [24] showed that in this case, the syntactic monoids of the languages defined are in the pseudovariety **DO** of monoids in which every regular $\mathcal{J}$-class (*i.e.,* every $\mathcal{J}$-class that contains an idempotent) is an *orthodox semigroup*–that is, a semigroup in which the product of two idempotents is idempotent. More precisely, they show:

**Theorem 3.2.** A language $L$ is definable by a two-variable sentence over $<$ in which no modular quantifier appears within the scope of an ordinary quantifier if and only if $M(L) \in \mathbf{DO}$ and every group in $M(L)$ is solvable.

Furthermore, $L$ is definable by such a sentence in which no modular quantifier appears within the scope of another quantifier if and only if $M(L) \in \mathbf{DO}$ and every group in $M(L)$ is abelian.

Let us illustrate this theorem with two examples. As already noted, our canonical example $(\sigma\tau)^*$ has a syntactic monoid in which the $\mathcal{J}$-class containing the idempotents $\sigma\tau$ and $\tau\sigma$ is not a subsemigroup. Thus this language cannot be defined by a two-variable sentence in which the modular quantifiers appear outside the ordinary quantifiers.

Second, consider the language consisting of words over $\{\sigma, \tau\}$, of the form $w\tau\sigma^k$, where $k \geq 0$, and $w$ contains an even number of occurrences of $\sigma$. This is defined by the sentence

$$\exists^{0 \bmod 2} x(Q_\sigma x \wedge \exists y(x < y \wedge Q_\tau y),$$

in which the modular quantifier appears outside the ordinary quantifier. The underlying set of the syntactic monoid $M$ is

$$\mathbf{Z}_2 \cup (\mathbf{Z}_2 \times \mathbf{Z}_2).$$

Words of the form $\sigma^i$ are map to the element $i \bmod 2$. Words of the form $w\tau\sigma^k$, where $w$ contains $j$ occurrences of $\sigma$ are mapped to $(j \bmod 2, k \bmod$

2). The multiplication in $M$ is given by

$$i \cdot j = (i + j) \bmod 2,$$

$$i \cdot (j, k) = ((i + j) \bmod 2, k),$$
$$(j, k) \cdot i = (j, (k + i) \bmod 2),$$
$$(j, k)(j', k') = ((j + k + j') \bmod 2, k').$$

The two $\mathcal{J}$-classes have underlying sets $\mathbf{Z}_2$ itself, and $(\mathbf{Z}_2 \times \mathbf{Z}_2)$ and the idempotents are 0,(0,0) and (1,1). Observe that this monoid is itself an orthodox semigroup.

Once again, there is a connection to computational complexity: Koucky, *et.al.* [11] show that the languages whose syntactic monoids are in **DO** and contain only abelian groups are precisely the regular languages recognized by $ACC^0$ circuits with only a linear number of wires.

## 4 Conclusion

Problems about the expressive power of modular quantifiers with unrestricted numerical predicates lie at the very edge of current knowledge about computational complexity. In all likeliehood, we are a long way from solving them. We have, however, been able to apply algebraic methods to obtain a thorough understanding of what happens when we use regular numerical predicates. This has led to large array of results that are deep and interesting in their own right, and provides valuable intuition about what is probably going on in the elusive general case.

## References

[1] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $NC^1$. *J.Comp. Syst. Sci.*, 38:150–164, 1989.

[2] D. Barrington, K. Compton, H. Straubing, and D. Thérien. Regular languages in $NC^1$. *J.Comp. Syst. Sci.*, 44:478–499, 1992.

[3] P. Beame, S. Cook, and J. Hoover. Log-depth circuits for division and related problems. *SIAM J. Computing*, 15:994–1003, 1986.

[4] J. Brzozowski and R. Knast. The dot-depth hierarchy of star-free languages is infinite. *J. Comp. Syst. Sci.*, 16:37–55, 1978.

[5] A. Chandra, L. Stockmeyer, and U. Vishkin. Constant-depth reducibility. *SIAM J. Computing*, 13:423–439, 1984.

[6] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. *J. Math Systems Theory*, 17:13–27, 1984.

[7] Y. Gurevich and H. Lewis. A logic for constant-depth circuits. *Information and Control*, 61:65–74, 1984.

[8] N. Immerman. Languages that capture complexity classes. *SIAM J. Computing*, 16:760–778, 1987.

[9] N. Immerman and D. Kozen. Definability with a bounded number of bound variables. *Information and Computation*, 83:121–13, 1989.

[10] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Berkeley, 1968.

[11] Michal Koucký, Pavel Pudlák, and Denis Thérien. Bounded-depth circuits: separating wires from gates. In *Proc. 37th ACM Symp. on Theory of Computing (STOC'05)*, pages 257–265, 2005.

[12] K. Krohn and J. Rhodes. The algebraic theory of machines I. *Trans. Amer. Math. Soc.*, 116:450–464, 1965.

[13] R. McNaughton and S. Papert. *Counter-Free Automata.* MIT Press, 1971.

[14] J. Rhodes and B. Tilson. The kernel of monoid morphisms. *J. Pure and Applied Algebra*, 62:227–268, 1989.

[15] A. Roy and H. Straubing. Definability of languages by generalized first-order formulas over (**N**,+). *SIAM J. Computing*, 37:502–521, 2007.

[16] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Computation*, 8:190–194, 1965.

[17] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. *Proc. 19th ACM STOC*, pages 77–82, 1987.

[18] H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theoret. Comput. Sci.*, 13:137–150, 1981.

[19] H. Straubing. *Finite Automata, Formal Logic and Circuit Complexity.* Birkhaüser, 1994.

[20] H. Straubing and D. Thérien. Weakly iterated block products of finite monoids. In *Proc. of the 5th Latin American Theoretical Informatics Conference (LATIN '02)*, 2002.

[21] H. Straubing and D. Thérien. Regular languages defined by generalized first-order formulas with a bounded number of bound variables. *Theory of Computing Systems*, 36:29–69, 2003.

[22] H. Straubing, W. Thomas, and D. Thérien. Regular languages defined with generalized quantifiers. *Proc. 15th ICALP, LNCS*, 317:561–575, 1988.

[23] H. Straubing, W. Thomas, and D. Thérien. Regular languages defined with generalized quantifiers. *Information and Computation*, 118:289–301, 1995.

[24] Pascal Tesson and Denis Thérien. Restricted two-variable sentences, circuits and communication complexity. In *Proc. 32nd Int. Conf. on Automata, Languages and Programming (ICALP'05)*, pages 526–538, 2005.

[25] 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.

[26] W. Thomas. Classifying regular events in symbolic logic. *J. Computer and System Sciences*, 25:360–376, 1982.

[27] W. Thomas. An application of the Ehrenfeucht-Fraïssé game in formal language theory. *Mem. Soc. Math. France*, 16:11–21, 1984.