Logical Methods in Computer Science
Vol. 3 (1:4) 2007, pp. 1–1–37
www.lmcs-online.org

Submitted Apr. 15, 2006
Published Feb. 23, 2007

LOGIC MEETS ALGEBRA: THE CASE OF REGULAR LANGUAGES *

PASCAL TESSON [a] AND DENIS THÉRIEN [b]

[a] Département d'Informatique et de Génie Logiciel, Université Laval
e-mail address: pascal.tesson@ift.ulaval.ca

[b] School of Computer Science, McGill University
e-mail address: denis@cs.mcgill.ca

Abstract. The study of finite automata and regular languages is a privileged meeting point of algebra and logic. Since the work of Büchi, regular languages have been classified according to their descriptive complexity, i.e. the type of logical formalism required to define them. The algebraic point of view on automata is an essential complement of this classification: by providing alternative, algebraic characterizations for the classes, it often yields the only opportunity for the design of algorithms that decide expressibility in some logical fragment.

We survey the existing results relating the expressibility of regular languages in logical fragments of monadic second order logic with successor with algebraic properties of their minimal automata. In particular, we show that many of the best known results in this area share the same underlying mechanics and rely on a very strong relation between logical substitutions and block-products of pseudovarieties of monoids. We also explain the impact of these connections on circuit complexity theory.

1. Introduction

Kleene's theorem insures that finite automata and regular expressions have the same expressive power and so we tend to forget that these two points of view on regular languages are of a different nature: regular expressions are well suited to reflect the combinatorial structure of a language while finite automata are first and foremost algebraic objects. It is intuitively clear that the combinatorial properties of a regular language should somehow be reflected in the structure of the corresponding automaton but this is difficult to formalize without resorting to algebra.

2000 ACM Subject Classification: D.3.1; F.1.1; F.1.3; F.4.1; F.4.3.

Key words and phrases: Descriptive complexity, regular languages, semigroup theory.

* The second author gave an invited lecture on this topic at the Seventh International Workshop on Logic and Computational Complexity (LCC'05) organized as a satellite workshop of the Logic in Computer Science Conference (LICS'05). We wish to thank the LCC organizers for that opportunity and for suggesting the present survey.

[a,b] The research of P. Tesson and D. Thérien is supported in part by the Natural Sciences and Engineering Council of Canada (NSERC) and the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT).

DOI:10.2168/LMCS-3 (1:4) 2007

On one hand, the algebraic point of view on finite automata pioneered by Eilenberg [Eil76] has been a driving force in our understanding of regular languages: each letter of an automaton's alphabet defines a transformation of the set of states and one can identify an automaton with its transition monoid, i.e. the finite monoid generated by these functions. Any regular language L can then be canonically associated with the transition monoid of its minimal automaton (the syntactic monoid of L) and many important classes of regular languages defined combinatorially can be characterized by the algebraic properties of their syntactic monoid.

On the other hand, Büchi showed in 1960 that the expressive power of monadic second order logic with successor $\mathbf{MSO}[\mathrm{S}]$ (or equivalently with order $\mathbf{MSO}[<]$) was exactly that of finite automata [Büc60]. Since then numerous results have related the expressive power of various sublogics to well-known classes of regular languages. The most notable example of this sort concerns languages definable by a first-order sentence using order. McNaughton and Papert showed that a regular language is definable in $\mathbf{FO}[<]$ if and only if it is star-free [MP71], i.e. if and only if the language can be described by a regular expression constructed from the letters of the alphabet, the empty set symbol, concatenation, union and complementation.

The result of McNaughton and Papert is non-trivial but it is stating an equivalence of two, not so different combinatorial descriptions of the class $\mathcal{SF}$ of star-free languages and neither of these is of any help to decide if a given language belongs to $\mathcal{SF}$. This is precisely why the algebraic point of view on automata is so fruitful. Intuitively, the fact that a language is star-free should translate into structural properties of the corresponding automaton and indeed, an earlier result of Schützenberger shows that $L \in \mathcal{SF}$ if and only if its syntactic monoid contains no non-trivial group [Sch65]. This immediately provides an algorithm to decide definability in $\mathbf{FO}[<]$. While it is fairly easy to show that a star-free language has a group-free syntactic monoid, the converse requires a very good understanding of the algebraic structure of these monoids.

Over the last thirty years, many of the most natural fragments of $\mathbf{MSO}[<]$, including a number of temporal sublogics of $\mathbf{LTL}$, have been characterized algebraically in the same way. It seems surprising, at first glance, that so many questions about the expressivity of logical fragments of $\mathbf{MSO}[<]$ have an algebraic answer but Straubing provided elements of a meta-explanation of the phenomenon [Str02]. For the majority of results simultaneously providing alternate logical, algebraic and combinatorial descriptions of a same class of regular languages, the greatest challenge is to establish the bridge between the combinatorial or logical characterization and the algebraic one.

One objective of the present survey is to give an overview of the existing results in this line of work and to provide examples illustrating the expressive power of various classes of logical sentences. We also want to demonstrate that our understanding of the underlying mechanics of the interaction between logic and algebra in this context has recently grown much deeper. We have a much more systematic view today of the techniques involved in bridging the algebraic and logical perspectives on regular languages and this seems primordial if we hope to extend them to more sophisticated contexts such as the theory of regular tree-languages.

We focus particularly on one such technique known as the block-product/substitution principle. Substitutions are a natural logical construct: informally, a substitution replaces the label predicates $Q_a x$ of an $\mathbf{MSO}[<]$ sentence ϕ by a formula with free variable x. We want to understand the extra expressive power afforded to a class of sentences Λ when we

substitute the label predicates of the ϕ in Λ by formulas from a class Γ. Under the right technical conditions, this logical operation can be put in correspondence with the block-product operation on pseudovarieties, an algebraic construct tied to bilateral semidirect products which reflects the combinatorial structure of substitutions. While this connection is not always as robust as we would hope it to be, it is still sufficient to derive the results of McNaughton-Papert and Schützenberger mentioned earlier, as well as results on temporal logic [CPP93, TW98, TW02, TW04], first-order sentences augmented with modular quantifiers [STT95] and sentences with a bounded number of variables [TW98, ST02, ST03, TT06].

Both logic and algebra have also contributed significantly in boolean circuit complexity. In particular, the circuit complexity classes AC^0, CC^0 and ACC^0 have interesting logical and algebraic characterizations: a language L lies in AC^0 if and only if it definable by an **FO** sentence using arbitrary numerical predicates [GL84, Imm87] if and only if it can be recognized by a polynomial-length *program* over a finite aperiodic monoid [BT88]. This makes it possible to attack questions of circuit complexity using either the logical (e.g. [Str94, Lib04, LMSV01, Lyn82a, Str92, RS06, KLPT06]) or the algebraic perspective (e.g. [BST90, BS95, BS99, Bou05, GRS05, MPT91, Thé94]). Furthermore, the recent results on the expressivity of two-variable logical sentences using order [EVW97, TW98, ST03] have found surprising connections with regular languages which can be recognized by bounded depth circuits that use only $O(n)$ gates or $O(n)$ wires [KLPT06, KPT05] and the communication complexity of regular languages [TT05a, TT05b].

We begin by reviewing in Sections 2 and 3 the bases of the logical and algebraic approach to the study of regular languages and introduce the block-product/substitution principle. We then consider two types of applications of this principle in Sections 4 and 5 and finally explore some of the connections to computational complexity in Section 6.

2. Logic on Words

We are interested in considering logical sentences describing properties of finite words in Σ^*. Variables in these sentences refer to positions in a finite word.

Example 2.1.
Consider for instance the sentence

$$\phi: \quad \exists x \exists y \forall z \quad \Big((x < y) \wedge Q_a x \wedge Q_a y \wedge [(x < z < y) \Rightarrow Q_c z] \Big) \ .$$

We think of ϕ as being true on words of $\{a,b,c\}^*$ that have positions x and y each holding the letter a so that any position in between them holds a c. We can therefore think of this sentence as defining the regular language $\{a,b,c\}^* a c^* a \{a,b,c\}^*$.

There is a considerable amount of literature dealing with the expressive power of these types of logics. Straubing's book on the links between logic, algebra and circuit theory [Str94] is certainly the reference which is closest in spirit to our discussion. Other valuable surveys and books include [Lib04, Pin96, Pin01, Tho97].

More formally we construct formulas using variables corresponding to positions in a finite word $w \in \Sigma^*$, usual existential and universal quantifiers, the boolean constants T and F, and boolean connectives. Moreover, for every letter $a \in \Sigma$, we have a unary predicate $Q_a x$ (the 'content' or 'label' predicate) which over a finite word w is interpreted as 'position x in the word w holds the letter a'. We further allow *numerical predicates* from some specified set $\mathcal{N} = \{R_1, \ldots, R_k\}$: the truth value of a numerical predicate $R_i(x_1, \ldots, x_{t_i})$

only depends on the values of the variables x_i and on the length[1] of the string w but not on the actual letters in those positions and we thus formally consider R_i of arity t_i as a subset of $\mathbb{N}^{t_i+1}$.

A *word structure* over alphabet Σ and variable set $\mathcal{V} = \{x_1, \ldots, x_k\}$ is a pair $(w, \overrightarrow{p})$ consisting of a word $w \in \Sigma^*$ and a list of pointers $\overrightarrow{p} = (p_1, \ldots, p_k)$ with $1 \leq p_i \leq |w|$ which associate each variable $x_i \in \mathcal{V}$ with a position p_i in the string. We identify the word w with the word structure $(w, \overrightarrow{0})$. Following [TW04], we further define a *pointed word* to be a word structure (w, p) with a single pointer p and a *pointed language* to be a set of pointed words. Alternatively, we can view a pointed word (w, p) as a triples $(u, a, v) \in \Sigma^* \times \Sigma \times \Sigma^*$ with $u = w_1 \ldots w_{p-1}$, $a = w_p$ and $v = w_{p+1} \ldots w_{|w|}$. Accordingly, we view pointed languages as subsets of $\Sigma^* \times \Sigma \times \Sigma^*$.

A *simple extension* of a word structure $(w, \overrightarrow{p})$ over $\Sigma, \mathcal{V}$ is a word structure $(w, \overrightarrow{p'})$ over $\Sigma, (\mathcal{V} \cup \{x_{k+1}\})$ such that $x_{k+1} \notin \mathcal{V}$ and $p_i = p'_i$ for $1 \leq i \leq k$. We can now formally define the semantics of our formulas in a natural way. If $w = w_1 \ldots w_t$ is a word and $\overrightarrow{p} = (p_1, \ldots, p_k)$ is a list of pointers to w, we have

$$
\begin{array}{lll}
(w, \overrightarrow{p}) \models Q_a x_i & \text{if} & w_{p_i} = a; \\
(w, \overrightarrow{p}) \models R_j(x_{i_1}, \ldots, x_{i_{t_j}}) & \text{if} & (p_{i_1}, \ldots, p_{i_{t_j}}, |w|) \in R_j; \\
(w, \overrightarrow{p}) \models \exists x_{k+1}(\phi(x_{k+1})) & \text{if} & \text{there exists a simple extension } (w, \overrightarrow{p'}) \text{ of } (w, \overrightarrow{p}) \\
 & & \text{such that } (w, \overrightarrow{p'}) \models \phi(x_{k+1}); \\
(w, \overrightarrow{p}) \models \forall x_{k+1}(\phi(x_{k+1})) & \text{if} & (w, \overrightarrow{p'}) \models \phi(x_{k+1}) \text{ for all simple extensions } (w, \overrightarrow{p'}).
\end{array}
$$

If ϕ is a sentence, i.e. a formula with no free variable, we denote as $L_\phi \subseteq \Sigma^*$ the language $L_\phi = \{w : (w, \overrightarrow{0}) \models \phi\}$. Similarly, formulas naturally define a set of word structures and it is often useful to consider the special case of formulas with a single free variable. Such a formula defines a set of pointed words (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 denote as P_ϕ the pointed language $P_\phi = \{(w, p) : (w, p) \models \phi\}$ and $\mathrm{P}(\Phi)$ the class of all P_ϕ with $\phi \in \Phi$.

For a set of numerical predicates $\mathcal{N}$, we denote as $\mathbf{FO}[\mathcal{N}]$ both the class of first-order sentences constructed with predicates in $\mathcal{N}$ and, with a slight abuse of notation, the class $\mathrm{L}(\mathbf{FO}[\mathcal{N}])$ of languages definable by such sentences. The expressive power of this logic is of course highly dependent on the choice of numerical predicates used. In particular, various results mentioned in our introduction can be combined to obtain:

Theorem 2.2. *A language L is*

- *definable in* $\mathbf{FO}[<]$ *if and only if L is a starfree regular language* [MP71]*;*
- *definable in* $\mathbf{FO}[*, +]$ *(addition and multiplication) if and only if L lies in the boolean circuit complexity class* DLOGTIME*-uniform* AC^0 [BIS90] *(see Section 6);*
- *definable in* $\mathbf{FO}$ *with no restriction on the class of numerical predicates used if and only if L lies in non-uniform* AC^0*.*

[1] Allowing the truth value of numerical predicates to also depend on the length of w might seem non-standard. It is equivalent to assuming that formulas have access to the constant max and since this constant is easily definable in first-order, it would appear that this relaxed definition of numerical predicates is unnecessary. However max cannot be defined in very weak fragments of $\mathbf{FO}$ or in logics using modular quantifiers. In these cases the connection of logic to circuit complexity (see Section 6) is best preserved with this slightly more general formulation.

There is a considerable body of work concerning the case where the available numerical predicates are order (¡), successor (S) or both. Of course $\mathbf{FO}[S]$ is contained in $\mathbf{FO}[<] = \mathbf{FO}[<, S]$ and that containment is known to be proper [Tho82]. In turn, $\mathbf{FO}[<]$ is clearly contained in $\mathbf{MSO}[S]$ and Büchi's theorem thus guarantees that all languages definable in these first-order fragments are regular.

One can further augment the expressive power of first-order sentences by introducing modular quantifiers $\exists^{i \bmod m} x\ \phi(x)$ (for some $m \geq 2$ and $i \leq m-1$). Intuitively $\exists^{i \bmod m} x\ \phi(x)$ holds true if property ϕ is true for i modulo m positions x. Formally

$$(w, \overrightarrow{p}) \models \exists^{i \bmod m} x_{k+1}\,(\phi(x_{k+1})) \qquad \text{if there exists } i \text{ modulo } m \text{ extensions } (w, \overrightarrow{p'}) \text{ of } (w, \overrightarrow{p}) \text{ such that } (w, \overrightarrow{p'}) \models \phi(x_{k+1}).$$

The next three examples will serve to illustrate results of the later sections.

Example 2.3.
The sentence

$$\exists^{0 \bmod 2} x\, \exists y\ \Big(Q_a x \wedge (y < x) \wedge Q_b y \wedge [\forall z\ ((y < z < x) \Rightarrow Q_c z)] \Big)$$

holds true for words over the alphabet $\Sigma = \{a, b, c, d\}$ in which there are an even number of positions x holding an a and whose prefix lies in $\Sigma^* bc^*$. The sentence thus defines the regular language

$$[(dc^*a \cup c \cup b)^* bc^* a(dc^*a \cup c \cup b)^* bc^* a]^* (dc^*a \cup c \cup b)^*.$$

Example 2.4.
The regular language $K = (b^*ab^*a)^*b\Sigma^*$ over the alphabet $\Sigma = \{a, b\}$ is defined by the sentence

$$\exists x\ \Big(Q_b x \wedge \exists^{0 \bmod 2} y\,[(y < x) \wedge Q_a y] \Big)\ .$$

Example 2.5.
The sentence

$$\exists x \forall y\ \Big(Q_a x \wedge [(y < x) \Rightarrow \neg Q_a y] \wedge \exists^{0 \bmod 2} z\ [(x < z) \wedge Q_c z] \Big)$$

is true of words over the alphabet $\{a, b, c\}$ such that the position x holding the first a has a suffix containing an even number of c's. Thus the language defined is

$$\{b, c\}^* a(\{a, b\}^* c\{a, b\}^* c\{a, b\}^*)^*.$$

We denote as $\mathbf{FO}[+]\mathbf{MOD}[\mathcal{N}]$ the class of first-order sentences constructed with the content predicates and numerical predicates in $\mathcal{N}$ and with existential, universal and modular quantifiers. We also denote as $\mathbf{MOD}[\mathcal{N}]$, the class of sentences in which only modular quantifiers are used. Once again Büchi's theorem guarantees that $\mathrm{L}(\mathbf{FO}[+]\mathbf{MOD}[<])$ contains only regular languages because the modular quantifiers can be simulated in monadic second order.

Definition 2.6. Let Σ be an alphabet and $\Phi = \{\phi_1(x), \ldots \phi_k(x)\}$ be a set formulas over Σ with at most[2] one free variable, say x. A Φ-substitution σ over Σ is a function mapping any sentence ψ over the alphabet 2^Φ (the power set of Φ) to a sentence $\sigma(\psi)$ over the alphabet Σ as follows. We assume without loss of generality that the set of variables used in ψ is disjoint from the set of variables in any ϕ_i and replace each occurrence of the predicate $Q_S y$ in ψ with $S \subseteq \Phi$ by the conjunction $\bigwedge_{\phi_i(x) \in S} \phi_i(y) \wedge \bigwedge_{\phi_i(x) \notin S} \neg \phi_i(y)$.

[2] It might be that some of the ϕ_i contain no occurrence of the free variable x and are thus sentences.

The following lemma formalizes the semantics of substitutions.

Lemma 2.7. *Let σ be a Φ-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 straightforward and is omitted [TW04, TT05b].

If Γ is a class of sentences and Λ is a class of formulas with one free variable we 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 Φ-substitution for some $\Phi \subseteq \Lambda$. Substitutions provide a natural way to decompose complex sentences into simpler parts. For instance, the class of $\mathbf{FO}[<]$ sentences of quantifier depth k can be decomposed as the class of sentences of depth 1 in which label predicates are replaced with formulas of quantifier depth $k-1$.

We are most interested in the case above where Γ is a class of sentences although the definition of $\Gamma \circ \Lambda$ can be naturally extended to the case where Γ is a class of formulas with one free variable. Under this more general setting the substitution operator is associative: if Γ, Λ, Ψ are classes of formulas with at most one free variable, then $\Gamma \circ (\Lambda \circ \Psi) = (\Gamma \circ \Lambda) \circ \Psi$.

3. Regular Languages, Finite Monoids and the Block Product/Substitution Principle

We give in the first half of this section a brief introduction to the algebraic theory of regular languages which is required for the sequel. A very thorough overview of the subject can be found in the survey of Pin [Pin97] or his earlier book [Pin86]. We also refer the interested reader to the survey of Weil which provides a shorter, more superficial introduction but considers more broadly the notion of algebraic recognizability for trees, infinite words, traces, pomsets and so on [Wei04]. In the section's second half, we state and prove the block-product/substitution principle which underlies many of the results presented in Sections 4 and 5.

3.1. Regular Languages, Automata and Finite Monoids.

A *semigroup* S is a set with a binary associative operation which we denote multiplicatively. A *monoid* M is a semigroup with a distinguished identity element 1_M. In the sequel, S and M always denote respectively a finite semigroup and a finite monoid. The set Σ^+ of finite non-empty words over Σ forms a semigroup under concatenation (the *free semigroup over* Σ) while the set Σ^* of finite words over Σ is a monoid with identity ϵ, the empty word.

We say that M divides the monoid N and write $M \prec N$ if M is the homomorphic image of a submonoid of N. A class $\mathbf{V}$ of finite monoids forms a *pseudovariety* if it is closed under finite direct product, homomorphic images and formation of submonoids. In particular, the following classes all form pseudovarieties:

- finite monoids $\mathbf{M}$;
- finite groups $\mathbf{G}$;
- finite solvable groups $\mathbf{G_{sol}}$;
- finite Abelian groups $\mathbf{Ab}$;
- finite solvable monoids $\mathbf{M_{sol}}$, i.e. monoids whose subgroups are solvable.

A monoid M is said to be *aperiodic* or *'group free'* if all its subgroups are trivial and we denote as $\mathbf{A}$ the pseudovariety of finite aperiodic monoids. The pseudovariety $\mathbf{SL}$ of semilattices consists of finite monoids which are idempotent ($x^2 = x$) and commutative ($xy = yx$) and it is easy to see that $\mathbf{SL} \subseteq \mathbf{A}$.

An element e of M is *idempotent* if $e^2 = e$. For any finite monoid, there is always an integer ω, the *exponent* of M such that x^ω is idempotent for all $x \in M$. Pseudovarieties can often be conveniently described[3] as the class of monoids satisfying a certain set of identities. For instance, the pseudovariety of groups $\mathbf{G}$ is the class of monoids satisfying $x^\omega y = yx^\omega = y$ (i.e. the only idempotent is the identity element of the group) and the pseudovariety $\mathbf{A}$ of aperiodics is defined by the identity $x^{\omega+1} = x^\omega$.

We say that the language $L \subseteq \Sigma^*$ is *recognized* by M if there exists a homomorphism $\rho : \Sigma^* \to M$ and a subset $F \subseteq M$ such that $L = \rho^{-1}(F)$. A simple variant of Kleene's theorem states that a language is regular if and only if it can be recognized by a finite monoid. When one chooses to consider languages as subsets of Σ^+ it is more natural to define recognition by finite semigroups and, for technical reasons, the algebraic theory of regular languages is slightly altered. The two parallel approaches coexist but cannot be completely reconciled despite their close relationship [Pin97]. For simplicity, we focus on the first case.

The *syntactic congruence* of a language $L \subseteq \Sigma^*$ is defined by setting $x \equiv_L y$ if and only if

$$uxv \in L \Leftrightarrow uyv \text{ for all } u, v \in \Sigma^*.$$

The Myhill-Nerode theorem states that $\equiv_L$ has finite index if and only if L is regular. The *syntactic monoid* $M(L)$ of L is the quotient $\Sigma^*/\equiv_L$ and is thus finite if and only if L is regular. It can be shown that $M(L)$ recognizes L and divides any monoid also recognizing L.

Example 3.1.
Consider the language $L = (ab)^*$. It is easy to see that for any word u containing two consecutive a or two consecutive b we have $u \notin L$ and, moreover, $xuy \notin L$ for any $x, y \in \{a, b\}^*$. Thus, any two such u are equivalent under the syntactic congruence and we denote the corresponding element of the syntactic monoid as 0 since it will satisfy $0m = m0 = 0$ for all monoid elements m. Simple computation shows that the syntactic monoid of L is the six-element monoid $B_2 = \{1, a, b, ab, ba, 0\}$ where multiplication is specified by $aba = a$, $bab = b$, $aa = bb = 0$. It is often convenient to name elements of a syntactic monoid using words in Σ^* that are minimal-length representatives for the different equivalence classes of the syntactic congruence.

For $u \in \Sigma^*$, the *right-quotient* of L by u is $Lu^{-1} = \{x : xu \in L\}$ and the left-quotient is defined symmetrically. A class $\mathcal{V}$ of languages is a variety of languages[4] if it is closed under boolean operations, left and right quotients and under inverse homomorphisms between free monoids (i.e. if $L \in \mathcal{V}$ and $\rho : \Gamma^* \to \Sigma^*$ is a homomorphism then $\rho^{-1}(L) \in \mathcal{V}$). The

[3] In fact, every pseudovariety has a possibly infinite set of defining *pseudo-identities* (see e.g. [Pin97] for a formal treatment).

[4] We should note that we are bypassing a technical yet important detail in our definition of varieties of languages. Strictly speaking, a variety of languages should not be defined as a set of languages but rather as an operator which assigns to each finite alphabet a set of languages over that alphabet. While that distinction is occasionally important in technical proofs, we prefer the slightly less formal description given here since it simplifies the presentation.

very tight relationship existing between varieties of languages and pseudovarieties of finite monoids is the cornerstone of algebraic automata theory.

Theorem 3.2 (Variety Theorem [Eil76])**.** *There is a natural bijection between pseudovarieties of finite monoids and varieties of languages: If* $\mathbf{V}$ *is a pseudovariety of finite monoids, then the class* $\mathrm{L}(\mathbf{V})$ *of regular languages recognized by some monoid in* $\mathbf{V}$ *forms a language variety.*

Conversely, if $\mathcal{V}$ *is a variety of languages then the pseudovariety of monoids* $\mathbf{V}$ *generated by the syntactic monoids of languages in* $\mathcal{V}$ *is such that* $\mathrm{L}(\mathbf{V}) = \mathcal{V}$*.*

One of the main objectives of algebraic automata theory is to explicitly relate natural varieties of regular languages with their algebraic counterpart or, conversely, describe combinatorially the variety of regular languages corresponding to a given pseudovariety of monoids. An algebraic characterization of a variety of languages $\mathcal{V}$ provides a natural approach for deciding if a given regular language L belongs to $\mathcal{V}$: checking if K belongs to $\mathcal{V}$ is equivalent to deciding if its syntactic monoid $M(K)$ belongs to the pseudovariety $\mathbf{V}$ such that $\mathrm{L}(\mathbf{V}) = \mathcal{V}$. The latter formulation of the problem is often easier to handle. In particular, all the pseudovarieties introduced thus far in this survey are such that determining membership of $M(K)$ in $\mathbf{V}$ amounts to checking that the monoid satisfies some *finite* set of defining identities (e.g. $x^{\omega+1} = x^\omega$ for the pseudovariety $\mathbf{A}$ of aperiodics). This requires an amount of time polynomial in $|M(K)|$. Although, $|M(K)|$ is in general exponentially larger than the size of the representation of K, the problem of testing whether $K \in \mathrm{L}(\mathbf{V})$ for a K specified by an automaton or a regular expression can be shown to lie in PSPACE for all pseudovarieties considered thus far. There are however pseudovarieties for which membership is undecidable and many problems in this line of work remain open [Alm94, Pin97].

The best-known instance of an algebraic characterization of a variety of languages is Schützenberger's theorem:

Theorem 3.3 ([Sch65])**.** *A regular language is star-free if and only if its syntactic monoid is group-free, i.e.* $\mathrm{L}(\mathbf{A}) = \mathcal{SF}$*.*

The theorem of McNaughton and Papert [MP71], whose proof we sketch in Section 4, further shows a language is star-free if and only if it is definable in $\mathbf{FO}[<]$.

Example 3.4.
A simple calculation shows that the syntactic monoid of the language $(ab)^*$, which we considered in Example 3.1, has exponent 2 and satisfies $x^3 = x^2$. It is therefore aperiodic and so there must exist a star-free expression and an $\mathbf{FO}[<]$ sentence defining $(ab)^*$. To construct a star-free expression, it suffices to note that $(ab)^*$ is the set of words starting with a, ending with b and having no consecutive a's or consecutive b's. Since the complement of the empty set $\emptyset^c$ is simply $\{a, b\}^*$, the following is a star-free expression defining $(ab)^*$:

$$a\emptyset^c \cap \emptyset^c b \cap (\emptyset^c aa\emptyset^c)^c \cap (\emptyset^c bb\emptyset^c)^c$$

The corresponding $\mathbf{FO}[<]$ sentence is

$$\forall x \forall y \Big((Q_b x \to [\exists z\ (z < x)]) \wedge (Q_a x \to [\exists z\ (x < z)]) \wedge$$
$$([((x \neq y) \wedge Q_a x \wedge Q_a y) \vee ((x \neq y) \wedge Q_b x \wedge Q_b y)] \to (\exists z\ [(x < z < y) \vee (y < z < x)]))\Big).$$

The notion of recognition of a language by a monoid can naturally be extended to pointed languages: we say that the pointed language $\dot{K} \subseteq \Sigma^* \times \Sigma \times \Sigma^*$ 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

$$\dot{K} = \{(w,p) : \big(h_l(w_1 \dots w_{p-1}), w_p, h_r(w_{p+1} \dots w_{|w|})\big) \in T\}.$$

For a pseudovariety $\mathbf{V}$ we denote as $\mathrm{P}(\mathbf{V})$ the set of pointed languages recognized by a monoid in $\mathbf{V}$. Abusing our terminology, it is convenient to think of ordinary words in Σ^* as pointed words with $p = 0$ and thus view $\mathrm{L}(\mathbf{V})$ as a subset of $\mathrm{P}(\mathbf{V})$. Note that $\mathrm{P}(\mathbf{V})$ is closed under boolean operations and inverse homomorphisms.

While relating star-freeness, aperiodicity and **FO**-definability is far from trivial, there are cases in which such three-way equivalences are easy to obtain and the following lemma is particularly useful in inductive arguments. Let $\mathbf{FO}_1[<]$ denote the class of first-order *sentences* with a single quantified variable and let $\mathbf{FOF}_1[<]$ denote the class of first-order *formulas* with a single quantified variable and at most one free variable. We similarly denote $\mathbf{MOD}_1[<]$ and $\mathbf{MODF}_1[<]$ the analog classes when ordinary quantifier quantifiers are replaced with modular ones.

Recall that **SL** and **Ab** respectively denote the pseudovarieties of semilattices and Abelian groups.

Lemma 3.5.

(1) $\mathrm{L}(\mathbf{SL})$ *is the Boolean algebra generated by languages of the form* $\Sigma^* a \Sigma^*$ *where* Σ *is a finite alphabet and* $a \in \Sigma$. *Furthermore* $\mathrm{L}(\mathbf{SL}) = \mathrm{L}(\mathbf{FO}_1[<])$ *and* $\mathrm{P}(\mathbf{SL}) = \mathrm{P}(\mathbf{FOF}_1[<])$.

(2) $\mathrm{L}(\mathbf{Ab})$ *is the Boolean algebra generated by languages of the form* $\{w : |w|_a \equiv i \pmod m\}$ *with* $i, m \in \mathbb{N}$. *Furthermore* $\mathrm{L}(\mathbf{Ab}) = \mathrm{L}(\mathbf{MOD}_1[<])$ *and* $\mathrm{P}(\mathbf{Ab}) = \mathrm{P}(\mathbf{MODF}_1[<])$.

Proof sketch. The syntactic monoid of the language $\Sigma^* a \Sigma^*$ of words containing an a is the two-element semilattice $\{1, 0\}$ with multiplication given by $x0 = 0x = 0$. Thus, any boolean combination of such languages can be recognized by a direct product of copies of this semilattice.

Conversely, if M is a semilattice and $\rho : \Sigma^* \to M$ is a homomorphism, then by commutativity and idempotency, the value of $\rho(w)$ only depends on the set of letters occurring in w. Thus, if $F \subseteq M$ then $\rho^{-1}(F)$ is in the boolean algebra generated by the $\Sigma^* a \Sigma^*$.

The language $\Sigma^* a \Sigma^*$ can be defined by the sentence $\exists x Q_a x$ and any $\mathbf{FO}_1$ sentence is a boolean combination of sentences of that form and therefore $\mathrm{L}(\mathbf{SL}) = \mathrm{L}(\mathbf{FO}_1[<])$. Similarly, $\mathbf{FOF}_1[<]$ formulas with free variable y and bound variable x are boolean combinations of sentences of the form $\exists x\ [(x * y) \wedge Q_a x]$ where $* \in \{<, >, =\}$ and one can conclude $\mathrm{P}(\mathbf{SL}) \subseteq \mathrm{P}(\mathbf{FOF}_1[<])$.

The case of Abelian groups is handled similarly: one can show that the variety of languages $\mathrm{L}(\mathbf{Ab})$ consists of languages L such that membership of a word w in L only depends on the number of occurrences of each letter in w modulo some integer m. □

The above lemma might give the impression that whenever $\mathbf{V}$ is a pseudovariety such that the class of languages $\mathrm{L}(\mathbf{V})$ has a meaningful logical description, then the class of pointed languages $\mathrm{P}(\mathbf{V})$ also has a meaningful (and closely related) logical description. This is unfortunately not the case and, in fact, there are very few classes Λ of formulas with one free variable whose expressive power can be characterized algebraically as $\mathrm{P}(\mathbf{V})$ for some pseudovariety $\mathbf{V}$.

3.2. **Block-Products and Substitutions.**

Let M and N be finite monoids. To distinguish the operation of M and N, we denote the operation of M as $+$ and its identity element as 0, although this operation is not necessarily commutative. A left-action of N on M is a function mapping pairs $(n, m) \in N \times M$ to $nm \in M$ and satisfying $n(m_1 + m_2) = nm_1 + nm_2$, $n_1(n_2m) = (n_1n_2)m$, $n0 = 0$ and $1m = m$. Given a left-action of N on M, the *semidirect product* $M \rtimes N$ (with respect to this action) is the monoid with elements in $M \times N$ and multiplication defined as $(m_1, n_1)(m_2, n_2) = (m_1 + n_1m_2, n_1n_2)$. It can be verified that this operation is indeed associative and that $(0, 1)$ acts as the identity element.

Right actions are defined symmetrically and naturally lead to the notion of *reverse semidirect products*. If we have both a right and a left-action of N on M that further satisfy $n_1(mn_2) = (n_1m)n_2$, we define the *bilateral semidirect product* $M{*}{*}N$ as the monoid with elements in $M \times N$ and multiplication defined as $(m_1, n_1)(m_2, n_2) = (m_1n_2 + n_1m_2, n_1n_2)$. This operation is associative and $(0, 1)$ acts as an identity for it. Semidirect products (resp. reverse semidirect) can then be viewed as the special case of bilateral semidirect products for which the right (resp. left) action on M is trivial. The *block product* of the pseudovarieties $\mathbf{V}, \mathbf{W}$, denoted $\mathbf{V} \Box \mathbf{W}$ is the pseudovariety generated by all bilateral semidirect products $M{*}{*}N$ with $M \in \mathbf{V}$, $N \in \mathbf{W}$.

(Bilateral) semidirect products are useful to decompose finite monoids of potentially complex structure into simpler components. For instance, it is well known that every finite group is isomorphic to an iterated semidirect product $G_1 \rtimes (G_2 \rtimes (\dots (G_{k-1} \rtimes G_k) \dots))$ where each G_i is a simple group and that a group is solvable if and only if there is such a decomposition in which all G_i are cyclic groups of prime order. For monoids which are not groups, the Krohn-Rhodes theorem [KR65] states that every finite monoid divides an iterated semidirect product (bracketed as above) where every term is either a simple group or the 'set/reset monoid' (or 'flip-flop') i.e. the three element monoid $\{1, s, r\}$ with multiplication satisfying $1x = x1 = x$, $xs = s$ and $xr = r$ for each x. The bilateral semidirect product allows decompositions with even simpler factors: every finite monoid divides an iterated bilateral semidirect product of the form

$$M_1{*}{*}(M_2{*}{*}(M_3{*}{*}(\dots M_{k-1}{*}{*}M_k)))$$

where each M_i is either a simple group or the two element semilattice [RT89].

Let $\mathbf{V_0}$ be the trivial pseudovariety (containing only the trivial monoid) and for $i \geq 0$ define inductively the pseudovarieties $\mathbf{V_{2i+1}} = \mathbf{G} \Box \mathbf{V_{2i}}$ and $\mathbf{V_{2i+2}} = \mathbf{SL} \Box \mathbf{V_{2i+1}}$. The last result stated in the previous paragraph implies in particular that the pseudovariety $\mathbf{M}$ of all finite monoids is the union of the $\mathbf{V_i}$ or, equivalently, that $\mathbf{M}$ is the smallest pseudovariety $\mathbf{W}$ satisfying $\mathbf{G} \Box \mathbf{W} = \mathbf{W}$ and $\mathbf{SL} \Box \mathbf{W} = \mathbf{W}$. The following theorem lists fundamental results that similarly decompose important pseudovarieties in terms of block-products. All results are either due to Rhodes and Tilson or can be inferred from their work [RT89].

Theorem 3.6.

(1) *The pseudovariety* $\mathbf{A}$ *of aperiodic monoids is the smallest pseudovariety satisfying* $\mathbf{SL} \Box \mathbf{A} = \mathbf{A}$.
(2) *The pseudovariety* $\mathbf{G_{sol}}$ *of solvable groups is the smallest pseudovariety satisfying* $\mathbf{Ab} \Box \mathbf{G_{sol}} = \mathbf{G_{sol}}$.
(3) *The pseudovariety* $\mathbf{M_{sol}}$ *of solvable monoids is the smallest pseudovariety satisfying* $\mathbf{Ab} \Box \mathbf{M_{sol}} = \mathbf{M_{sol}}$ *and* $\mathbf{SL} \Box \mathbf{M_{sol}} = \mathbf{M_{sol}}$.

The block product operation on pseudovarieties is not associative: it can be shown that $(\mathbf{U} \,\square\, \mathbf{V}) \,\square\, \mathbf{W} \subseteq \mathbf{U} \,\square\, (\mathbf{V} \,\square\, \mathbf{W})$ but this inclusion is strict in general. The block product is mostly used as a means of decomposing large and complex pseudovarieties into smaller, simpler ones and the most classical applications of iterated block-products have relied on the stronger right-to-left bracketing. Theorem 3.6 for instance states that the pseudovariety of aperiodics is the union of all pseudovarieties of the form

$$\mathbf{SL} \,\square\, (\mathbf{SL} \,\square\, (\ldots \,\square\, (\mathbf{SL} \,\square\, \mathbf{SL}) \ldots)).$$

In Section 5 we show the relevance of the weaker left-to-right bracketing of iterated block-products when analyzing the expressive power of two-variable sentences.

The languages recognized by $\mathbf{V} \,\square\, \mathbf{W}$ 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 $w = w_1 \ldots w_n \in \Sigma^*$ we set

$$\tau(w) = \tau(w_1)\tau(w_2)\ldots\tau(w_n)$$

with

$$\tau(w_i) = (h_l(w_1 \ldots w_{i-1}), w_i, h_r(w_{i+1} \ldots w_n)) .$$

For a language $K \subseteq (N \times \Sigma \times N)^*$, let $\tau^{-1}(K) = \{w \in \Sigma^* : \tau(w) \in K\}$.

Theorem 3.7. [Str94, Pin97] *A regular language lies in* $\mathrm{L}(\mathbf{V} \square \mathbf{W})$ *iff it is the Boolean combination of languages in* $\mathrm{L}(\mathbf{W})$ *and languages* $\tau^{-1}(K)$ *for some* $K \in \mathbf{V}$ *and* N*-transduction* τ *with* $N \in \mathbf{W}$*.*

Proof sketch. The proof is too technical to present in full detail but we give a brief overview of the main idea for completeness. The argument relies on the very definition of multiplication in the bilateral semidirect product $M * * N$. Recall that $(m_1, n_1)(m_2, n_2) = (m_1 n_2 + n_1 m_2, n_1 n_2)$ and so, by extension, if (m_1, n_1), $(m_2, n_2), \ldots, (m_t, n_t)$ are elements of $M * * N$ then their product $(m_1, n_1) \cdot \cdots \cdot (m_t, n_t)$ in $M * * N$ is given by

$$(m_1 n_2 n_3 \ldots n_t + n_1 m_2 n_3 \ldots n_t + \ldots + n_1 \ldots n_{t-2} m_{t-1} n_t + n_1 \ldots n_{t-1} m_t, n_1 \ldots n_t).$$

Fix an element in $(m, n) \in M * * N$ and consider the language $E_{(m,n)} \subseteq (M * * N)^*$ consisting of finite sequences $(m_1, n_1), \ldots, (m_t, n_t)$ of elements of $M * * N$ that multiply out to (m, n). Similarly, let $E_{(m,*)}$ be the union over all n of all $E_{(m,n)}$ and let $E_{(*,n)}$ be the union over all m of the $E_{(m,n)}$: we thus have $E_{(m,n)} = E_{(m,*)} \cap E_{(*,n)}$. Finally denote by $E_m \subseteq M^*$ the set of words of M^* that multiply out to m in M. Let τ be the N-transduction which maps $w \in (M * * N)^*$ to $\tau(w) = \tau(w_1) \ldots \tau(w_{|w|})$ with $\tau(w_i) = (n_1 \ldots n_{i-1}, m_i, n_{i+1} \ldots n_t)$. If we identify these triples with the element $n_1 \ldots n_{i-1} m_i n_{i+1} \ldots n_t$ of M then by the above expression we obtain immediately that $E_{(m,*)}$ is $\tau^{-1}(E_m)$. Note that if $M \in \mathbf{V}$ then $E_m \in \mathrm{L}(\mathbf{V})$. On the other hand it is easy to see that if $N \in \mathbf{W}$ then $E_{(*,n)} \in \mathrm{L}(\mathbf{W})$. This argument in fact suffices to establish that every language in $\mathrm{L}(\mathbf{V} \,\square\, \mathbf{W})$ is a Boolean combination of languages in $\mathrm{L}(\mathbf{W})$ and languages $\tau^{-1}(K)$ for some $K \in \mathbf{V}$ and N-transduction τ with $N \in \mathbf{W}$.

The converse statement, while more involved technically, proceeds along the same lines. □

There is a striking similarity between the notion of transduction and that of substitution discussed in the previous section. We formalize this crucial correspondence in the next lemma which we refer too as the *block-product/substitution principle*. Thérien and Wilke [TW04] were the first to use this specific terminology although it is fair to say that the idea was implicitly present in the work of Straubing [Str94]. In [TW04], the lemma is stated for temporal logics. The formulation given here is taken from [TT05b].

Lemma 3.8 (Block-product/subsitution principle)**.**
Let Γ *be a class of* $\mathbf{FO}+\mathbf{MOD}[<]$ *sentences and* Λ *a class of* $\mathbf{FO}+\mathbf{MOD}[<]$ *formulas with one free variable. If* $\mathbf{V}, \mathbf{W}$ *are pseudovarieties of finite monoids such that* $\mathrm{L}(\Gamma) = \mathrm{L}(\mathbf{V})$ *and* $\mathrm{P}(\Lambda) = \mathrm{P}(\mathbf{W})$*, then* $\mathrm{L}(\Gamma \circ \Lambda) = \mathrm{L}(\mathbf{V} \,\square\, \mathbf{W})$*.*

Proof. Since $\mathrm{L}(\mathbf{V} \,\square\, \mathbf{W})$ 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 $\mathrm{L}_{\sigma(\psi)} \in \mathrm{L}(\mathbf{V} \,\square\, \mathbf{W})$. Let w be some word in Σ^* and $\Phi = \{\phi_1, \dots \phi_k\}$ be the formulas used by σ. Since $\mathrm{P}(\mathbf{W}) = \mathrm{P}(\Lambda)$, the pointed languages $P_{\phi_j} : \{(w,i) : (w,i) \models \phi_j\}$ can be recognized by monoids $N_1, \dots, N_k$ in $\mathbf{W}$ and $N = N_1 \times \dots \times N_k$ recognizes any Boolean combination of them. This implies the existence of two morphisms $h_l, h_r : \Sigma^* \to N$ such that the membership of a pointed word (w,i) in each N_j can be determined by the value of the triple $(h_l(w_1 \dots w_{i-1}), w_i, h_r(w_{i+1} \dots w_n))$. Using these two homomorphisms, we therefore obtain an N-transduction τ such that for each i, the value of $\tau(w_i)$ is sufficient to determine the set $\{\phi_j : (w,i) \models \phi_j\}$. Since we assume that L_ψ is recognized by a monoid M in $\mathbf{V}$, we get that $\mathrm{L}_{\sigma(\psi)} = \tau^{-1}(K)$ for some $K \subseteq (N \times \Sigma \times N)^*$ also recognized by M. Hence, by Theorem 3.7, $\mathrm{L}_{\sigma(\psi)} \in \mathrm{L}(\mathbf{V} \,\square\, \mathbf{W})$.

For the right-to-left containment, we need to show that any language of $\mathrm{L}(\mathbf{V} \,\square\, \mathbf{W})$ can be described by a sentence of $\Gamma \circ \Lambda$ and we proceed similarly. If τ is an N-transduction for some $N \in \mathbf{W}$ then for any triple $(n_1, a, n_2) \in N \times \Sigma \times N$, the pointed language

$$T_{(n_1,a,n_2)} = \{(w,i) : \tau(w_i) = (n_1, a, n_2)\}$$

is in $\mathrm{P}(\mathbf{W})$ and is thus definable by some formula $\phi_{(n_1,a,n_2)}$ in $\mathrm{P}(\Lambda)$. Consider the substitution σ defined by these ϕ_{n_1,a,n_2} and note that for any $w \in \Sigma^*$ and any position $1 \leq i \leq |w|$, *exactly one* of the $\phi_{(n_1,a,n_2)}$ is true at i. Hence $\sigma^{-1}(w_i) = \{\phi_{(n_1,a,n_2)} | (w,i) \models \phi_{(n_1,a,n_2)}\}$ is always a singleton and the range of possible values can be identified with the set $N \times \Sigma \times N$. Any language $K \subseteq (N \times \Sigma \times N)^*$ in $\mathrm{L}(\mathbf{V})$ 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 σ substitution. □

Many results giving algebraic characterizations of regular languages defined in a fragment Λ of $\mathbf{FO}+\mathbf{MOD}[<]$ more or less explicitly rely on some form of this lemma. It is often rather easy to characterize algebraically the expressivity of very weak fragments of $\mathbf{FO}+\mathbf{MOD}[<]$ (e.g. Lemma 3.5). Furthermore, sufficiently robust classes Λ can typically be decomposed through iterated substitutions of these weak fragments. Applying the block-product/substitution principle we are thus able to characterize the expressive power of Λ by analyzing an iterated block-product.

For a number of reasons, however, this general paradigm cannot be applied too generally.

- Straubing showed that the class of regular languages definable in the most natural fragments of $\mathbf{MSO}[<]$ (in particular, fragments of first-order defined by quantifier type, quantifier alternation, quantifier depth, number of variables and so on) are all

$\mathcal{C}_{lm}$-varieties of languages (see [Str02] for a formal definition) and are varieties of languages in the sense we defined earlier when we consider subclasses of $\mathbf{FO}^+\mathbf{MOD}[<]$. Thus the expressive power of these fragments have *some* algebraic characterization. Preliminary investigations unfortunately indicate that classes of pointed languages definable in similar fragments only rarely correspond to $\mathrm{P}(\mathbf{V})$ for some pseudovariety $\mathbf{V}$, as we noted after Lemma 3.5. This state of affairs limits the possible range of applications of the block-product/substitution principle.

- We mentioned in Section 2 that the substitution operator is associative and the block-product/subsitution principle might lead one to find this fact in apparent contradiction with the non-associativity of the block-product. If Γ is a class of sentences and Λ, Φ are classes of formulas with at most one free variable, then indeed we have $\Gamma \circ (\Lambda \circ \Phi) = (\Gamma \circ \Lambda) \circ \Phi$. Suppose that $\mathbf{U}, \mathbf{V}, \mathbf{W}$ are pseudovarieties such that $\mathrm{L}(\Gamma) = \mathrm{L}(\mathbf{U})$, $\mathrm{P}(\Lambda) = \mathrm{P}(\mathbf{V})$ and $\mathrm{P}(\Phi) = \mathrm{P}(\mathbf{W})$ then the principle insures us first that $\mathrm{L}(\Gamma \circ \Lambda) = \mathrm{L}(\mathbf{U} \,\square\, \mathbf{V})$ and, with a second application, that $\mathrm{L}(\Gamma \circ (\Lambda \circ \Phi)) = \mathrm{L}((\mathbf{U} \,\square\, \mathbf{V}) \,\square\, \mathbf{W})$. However, we cannot in general infer $\mathrm{P}(\Lambda \circ \Phi) = \mathrm{P}(\mathbf{V} \,\square\, \mathbf{W})$.

4. Classical Results from the Block-Product/Subsitution Principle

4.1. Quantifier Depth.

Let us first see how the block-product/subsitution principle can provide a proof of McNaughton and Papert's characterization of $\mathbf{FO}[<]$ and Straubing, Thérien and Thomas' characterization of $\mathbf{FO}^+\mathbf{MOD}[<]$. For a sentence ψ, and a variable x not occurring in ψ, let $\psi_{[<x]}$ and $\psi_{[>x]}$ respectively denote the formulas obtained from ψ by restricting the scope of any quantified variable of ψ to values respectively strictly less than x and strictly greater than x. We rely on the following lemma.

Lemma 4.1 ([STT95])**.** *Any* $\mathbf{FO}^+\mathbf{MOD}[<]$ *formula* $\phi(x)$ *with a single free variable* x *can be rewritten as a boolean combination of formulas of the form* $Q_a x \wedge \rho_{[<x]} \wedge \chi_{[>x]}$ *such that the formulas* ρ *and* χ *have the same quantifier signature as* ϕ *i.e. their quantifier depths are equal and the order in which different types of quantifier types are nested (existential, universal, mod* m *counting) are the same.*

The proof is not conceptually difficult. By renaming variables we can assume that $\phi(x)$ does not contain any bound occurrence of x and this is the starting point of the construction. This rather trivial observation does not necessarily hold, however, when the sentences considered are only allowed a bounded number of variables as in Section 5.

Let us first focus on the problem of definability in $\mathbf{FO}[<]$. Let $\mathbf{SD^k}$ and $\mathbf{FD^k}$ respectively denote the class of $\mathbf{FO}[<]$ *sentences* of quantifier depth k and $\mathbf{FO}[<]$ *formulas* of quantifier depth k with at most one free variable. By definition of quantifier depth, we have $\mathbf{SD^{k+1}} = \mathbf{SD^1} \circ \mathbf{FD^k}$.

Theorem 4.2. *Let* $\mathbf{V_1} = \mathbf{SL}$ *and* $\mathbf{V_{k+1}} = \mathbf{SL} \,\square\, \mathbf{V_k}$*. A regular language* L *can be defined by an* $\mathbf{FO}[<]$ *sentence of quantifier depth* k *if and only if its syntactic monoid* M *lies in* $\mathbf{V_k}$*. In other words,* $\mathrm{L}(\mathbf{SD^k}) = \mathrm{L}(\mathbf{V_k})$*.*

Proof sketch. We argue by induction on k. The base case is provided by Lemma 3.5. For the induction step we use the fact that $\mathbf{SD^{k+1}} = \mathbf{SD^1} \circ \mathbf{FD^k}$. Since we know that $\mathrm{L}(\mathbf{SD^1}) = \mathrm{L}(\mathbf{SL})$ our inductive claim follows from the block-product/substitution principle

if we can show that $\mathrm{P}(\mathbf{FD^k}) = \mathrm{P}(\mathbf{V_k})$. This is precisely what Lemma 4.1 allows: any formula $\phi(x) \in \mathbf{FD^k}$ can be rewritten as a boolean combination of formulas of the form $Q_a x \wedge \rho_{[<x]} \wedge \chi_{[>x]}$ where ρ and χ are sentences of $\mathbf{SD^k}$. □

Membership of M in any individual $\mathbf{V_k}$ is trivially decidable because these pseudovarieties are *effectively locally finite* [Pin97]: for each t, k we can effectively construct a monoid $M_{t,k}$ such that any monoid $M \in \mathbf{V_k}$ with at most t generators is a divisor of $M_{t,k}$. On the logical side, this is essentially equivalent to noting that over a given alphabet there are only finitely many equivalent first-order sentences of any fixed quantifier depth. The decidability of individual $\mathbf{V_k}$ is in itself of moderate interest since one is typically not so interested in determining whether L is definable in some specific quantifier depth but rather whether L is $\mathbf{FO}[<]$-definable at all. In algebraic terms, we are more interested in deciding membership of M in the union of the $\mathbf{V_k}$ than in some specific $\mathbf{V_k}$. By part (1) of Theorem 3.6 we know that $\bigcup \mathbf{V_k} = \mathbf{A}$ and this provides the missing link to obtain the theorem of McNaughton and Papert which we combine with Schützenberger's Theorem to obtain:

Corollary 4.3. $\mathrm{L}(\mathbf{FO}[<]) = \mathrm{L}(\mathbf{A}) = \mathcal{S}F$.

Thus, deciding if a language K is $\mathbf{FO}[<]$ definable is equivalent to testing if K's syntactic monoid is aperiodic. The latter problem is clearly decidable and is in fact PSPACE-complete when K is specified by a finite automaton [CH91].

The same proof methods also yield an algebraic characterization of languages definable in $\mathbf{FO}^+\mathbf{MOD}[<]$ and $\mathbf{MOD}[<]$.

Theorem 4.4. *A language L is $\mathbf{FO}^+\mathbf{MOD}[<]$-definable if and only if its syntactic monoid M is solvable. Furthermore, L is $\mathbf{MOD}[<]$-definable if and only if M is a solvable group.*

Proof sketch. By part (2) of Lemma 3.5 we have $\mathrm{L}(\mathbf{Ab}) = \mathrm{L}(\mathbf{MOD}_1[<])$ as well as $\mathrm{P}(\mathbf{Ab}) = \mathrm{P}(\mathbf{MODF}_1[<])$. Using the inductive argument of Theorem 4.2 we get that the class of languages definable in $\mathbf{MOD}[<]$ (resp. $\mathbf{FO}^+\mathbf{MOD}[<]$) are those with syntactic monoids in the smallest pseudovariety $\mathbf{V}$ satisfying $\mathbf{Ab} \,\square\, \mathbf{V} = \mathbf{V}$ (resp. $\mathbf{Ab} \,\square\, \mathbf{V} = \mathbf{V}$ and $\mathbf{SL} \,\square\, \mathbf{V} = \mathbf{V}$). Theorem 3.6 completes the argument. □

The theorem immediately provides an algorithm to decide expressibility in these two logics. The result can be specialized to characterize the expressive power of $\mathbf{FO}^+\mathbf{MOD}[<]$ and of $\mathbf{MOD}[<]$ sentences when the modular quantifiers are restricted to specific moduli [BIS90, Str94, STT95].

4.2. **Quantifier Alternation.**

Quantifier depth is only one of many possible parameterizations of languages definable in $\mathbf{FO}[<]$. In particular it is natural to consider the hierarchy of first-order sentences defined by quantifier alternation. The block-product/substitution principle seems to be of no use in that case but the question can nevertheless be studied with algebraic and combinatorial perspectives. There is a natural parametrization of star-free languages in terms of concatenation depth. The *Straubing-Thérien hierarchy* is defined inductively as follows: a language over Σ^* has depth 0 if and only if it is $\emptyset$ or Σ^*. Level $k + 1/2$ of the hierarchy consists of unions of languages of the form $L_0 a_1 L_1 a_2 \dots a_t L_t$ where the a_i are letters of Σ and the L_i's are languages of depth k. Finally the $(k+1)$st level of the hierarchy is the boolean closure of the level $k + 1/2$. This hierarchy is closely related to the Brzozowski-Cohen or dot-depth hierarchy [CB71] (the precise correspondence was

established by Straubing [Str85, Pin97]). The Straubing-Thérien hierarchy is known to be infinite and the union of all its levels clearly corresponds to the class $\mathcal{SF}$ of star-free languages.

As usual, let $\Sigma_k[<]$ and $\Pi_k[<]$ denote the subclasses of $\mathbf{FO}[<]$ sentences defined by quantifier alternation.

Theorem 4.5 ([Tho82, PP86])**.** *A language L is definable in $\Sigma_k[<]$ if and only if L belongs to level $k + 1/2$ of the Straubing-Thérien hierarchy.*

In fact, the original result of Thomas [Tho82] relates the levels of the Brzozowski-Cohen dot-depth hierarchy with definability in $\Sigma_k[<, S]$ but the argument can be easily be adapted to obtain the theorem just stated [PP86]. The 'if' part of the theorem is immediate from the definition of the Straubing-Thérien hierarchy. Thomas' argument for the second half of the theorem does not involve any algebra and relies instead on Ehrenfeucht-Fraïssé games. In particular it provides a way to relate $\mathbf{FO}[<]$-definability and star-freeness without resorting to algebra (this is also true of [MP71]).

It is not hard to show that the kth levels of the Straubing-Thérien hierarchy are closed under inverse homomorphic images, left and right quotients, union and complementation and thus form varieties of languages. The variety theorem therefore guarantees that these classes correspond to some pseudovariety of finite monoids. Note in contrast that the $k+1/2$ levels do not form varieties of languages since they are not closed under complementation. They still are closed under inverse homomorphic images, quotients, union and intersection and therefore form what are known as *positive varieties* of languages. These can also be analyzed from an algebraic perspective using *ordered syntactic monoids* and *pseudovarieties of ordered monoids* [Pin86, Pin97]. The decidability of levels 1/2 and 1 of the Straubing-Thérien hierarchy follow from Simon's theorem on piecewise-testable languages [Sim75] and later refinements [Pin95, Pin97]. Level 3/2 is also decidable but considerable work is needed to establish this deep fact [PW97] (see [GS00] for an independent proof of the decidability of level 3/2 of the dot-depth hierarchy) and the decidability of level 2 is one of the most important open problems in algebraic automata theory [GS01, Pin97, PS81, PW97, PW01, Str88, SW92, Wei89].

There is in fact a general lesson to be learned from Theorem 4.5. We argued in the first half of this section that when Φ is a class of sentences and $\mathbf{V}$ is a pseudovariety such that $\mathrm{L}(\Phi) = \mathrm{L}(\mathbf{V})$ then the class Γ of sentences which are boolean combinations of sentences of the form $\exists x\ [Q_a x \wedge \psi_{[<x]} \wedge \chi_{[>x]}]$ with $\psi, \chi \in \Phi$ is such that $\mathrm{L}(\Gamma) = \mathrm{L}(\mathbf{SL} \,\square\, \mathbf{V})$. Clearly, the languages in $\mathrm{L}(\Gamma)$ are boolean combinations of languages of the form $L_1 a L_2$ with $L_1, L_2 \in \mathrm{L}(\Phi)$. These facts provide us with a bridge linking, under the correct technical assumptions, the logical operation of adding an extra existential quantifier, the algebraic operation of forming a block-product $\mathbf{SL} \,\square\, \mathbf{V}$ and the combinatorial operation of concatenation of two languages. The same idea can be extended to obtain combinatorial and algebraic counterparts to the addition of a whole block of existential quantifiers $\exists x_1 \ldots \exists x_k\ \phi(x_1, \ldots, x_k)$ on the logical side.

For a variety of languages $\mathcal{V}$, we define $\mathrm{Pol}(\mathcal{V})$ to be the class[5] of languages which are unions of languages of the form $L_0 a_1 L_1 \ldots a_k L_k$ for $L_i \in \mathcal{V}$. One can put in correspondence the *logical operation* of adding a block of existential quantification with the *combinatorial operator* of polynomial closure $\mathrm{Pol}(\mathcal{V})$ on varieties of languages. In other words, under some

[5] Note that in general $\mathrm{Pol}(\mathcal{V})$ is not a variety of languages because it need not be closed under complement. It does however form a positive variety in the sense of [Pin86, Pin97]

technical conditions, one can show that if Φ is a class of sentences and $\mathcal{V}$ is a language variety such that $\mathrm{L}(\Phi) = \mathcal{V}$ then a language K belongs to $\mathrm{Pol}(\mathcal{V})$ if and only if it can be defined as a positive boolean combination of sentences of the form

$$\exists x_1 \ldots \exists x_k \, [(x_1 < \ldots < x_k) \wedge Q_{a_1} x_1 \wedge \ldots \wedge Q_{a_k} x_k \wedge \psi^0_{[<x_1]} \wedge \psi^1_{[>x_1,<x_2]} \wedge \ldots \wedge \psi^k_{[>x_k]}]$$

where each ψ^j is a formula in Φ and where the subscript $\psi^j_{[>x_j,<x_{j+1}]}$ is the formula obtained from ψ^j by restricting the quantified variables to lie between x_j and x_{j+1}.

In turn the operator $\mathrm{Pol}(\mathcal{V})$ on varieties of languages is linked to an *algebraic operation* on pseudovarieties of monoids defined in terms of so-called *Mal'cev products* [PW97]. Similarly, the addition of a block of modular quantifiers is related to the closure of a variety of languages under products with counters which can also be described algebraically through Mal'cev products [Wei92].

4.3. **Sentences with Regular Predicates.**

The numerical predicate successor (S) is definable in $\mathbf{FO}[<]$ and so the expressive power of $\mathbf{FO}[<, S]$ (resp. $\mathbf{FO}{+}\mathbf{MOD}[<, S]$) is exactly that of $\mathbf{FO}[<]$ (resp. $\mathbf{FO}{+}\mathbf{MOD}[<]$).

The cases of $\mathbf{MOD}[<, S]$ and of the different $\Sigma_k[<, S]$, however, are more subtle: the algebraic characterization of languages definable in these fragments [Str94, STT95] would require the introduction of the notions of *syntactic semigroup*, semigroup pseudovarieties and $+$*−varieties of languages* which we chose to omit. Still, the fundamental tools of the analysis are conceptually very similar to the ones we presented in this section.

If successor is the only available numerical predicate, then the expressive power of first-order sentences is dramatically reduced. Thomas and later Straubing gave combinatorial and algebraic descriptions (the latter, again, in terms of syntactic semigroups) of the languages definable in $\mathbf{FO}[S]$ and showed that these form a strict subclass of the star-free regular languages [Str94, Tho82, Tho97]. The work of Thérien and Weiss [TW85] establishes the decidability of this class. The cases $\mathbf{MOD}[S]$ and $\mathbf{FO}{+}\mathbf{MOD}[S]$ are also investigated in Chapter VI of Straubing's book [Str94].

The extra expressive power afforded by the unary predicate $\equiv_{i,m} x$ (which is true at x if $x \equiv i \pmod m$) has also been considered [CPS06, Str02]. More generally, a numerical predicate $R \subseteq \mathbb{N}^t$ is said to be *regular* if it is definable in $\mathbf{FO}{+}\mathbf{MOD}[<]$. Equivalently, R is regular if it is definable in $\mathbf{FO}[<, \{\equiv_{i,m}\}]$ (see [Pél92]) and the terminology comes from yet another equivalent definition of regular predicates using finite automata [Str94]. Let Reg denote the class of regular numerical predicates: it follows from our definition that $\mathbf{FO}{+}\mathbf{MOD}[Reg] \subseteq \mathbf{MSO}[<]$ so this class consists only of regular languages and in fact regular predicates form the largest class of numerical predicates with this property [Pél92]. By definition, $\mathbf{FO}{+}\mathbf{MOD}[Reg] = \mathbf{FO}{+}\mathbf{MOD}[<]$ and the expressive power of the fragments $\mathbf{FO}[Reg]$, $\mathbf{MOD}[Reg]$ (among others) can be characterized algebraically [BCST92, STT93, Str94, Pél92].

5. Two-Variable Sentences and Temporal Logic

In the previous section, the application of the block-product/substitution principle was particularly fruitful because of the decomposition of the pseudovarieties $\mathbf{A}, \mathbf{G_{sol}}, \mathbf{M_{sol}}$ in terms of iterated block products of semilattices and Abelian groups (Theorem 3.6). As

we noted these iterated block products use the strong, right-to-left bracketing whereas the present section relies on decompositions using the weaker left-to-right bracketing.

5.1. **Sentences with a Bounded Number of Variables.**

It is common practice to construct logical sentences in such a way that any subformula $\phi(x)$ with a free variable x never contains an occurrence of x which is bound by a quantifier. This certainly avoids possible confusions although it is quite possible to construct sentences that do not obey this rule and still get unambiguous semantics by interpreting a variable as bound by the previous quantifier[6]. We illustrate this in the following two examples:

Example 5.1.
The three variable sentence of Example 2.5:

$$\exists x \forall y \left(Q_a x \wedge [(y < x) \Rightarrow \neg Q_a y] \wedge \exists^{0 \bmod 2} z \, [(x < z) \wedge Q_c z] \right)$$

can clearly be rewritten as the two-variable sentence

$$\exists x \forall y \left(Q_a x \wedge [(y < x) \Rightarrow \neg Q_a y] \wedge \exists^{0 \bmod 2} y \, [(x < y) \wedge Q_c y] \right).$$

In many cases, the rewriting is not as trivial.

Example 5.2.
We claim that the following **FO**[<] sentence can also be rewritten using only two variables.

$$\exists x \forall y \exists z \left(Q_a x \wedge [(x < y) \Rightarrow \neg Q_a y] \wedge Q_d z \wedge (x < z) \wedge [(x < y < z) \Rightarrow Q_c y] \right).$$

This sentence is true for words over $\Sigma = \{a, b, c, d\}$ in which there exists a position x that holds the last occurrence of a and whose suffix begins with some c's (possibly none) followed by a d. Thus the sentence defines the language $\Sigma^* a c^* d \{b, c, d\}^*$.

We claim that the following two-variable sentence defines the very same language.

$$\exists x \left(Q_a x \wedge [\forall y \, ((x < y) \Rightarrow \neg Q_a y)] \wedge \right.$$
$$\left. \exists y \, [(x < y) \wedge Q_d y \wedge \forall x \, [((x < y) \wedge \neg Q_c x) \Rightarrow (\exists y \, [(x \leq y) \wedge Q_a y])]] \right).$$

The first part of this second sentence also identifies x as the location of the last a. To understand how the rest of the sentence imposes the condition on the suffix of this position, it is more convenient to look first at the meaning of the most deeply nested subformulas and work back towards the outermost quantifiers: the most deeply nested subformula

$$\phi(x) : \exists y \, [(x \leq y) \wedge Q_a y)]$$

with free variable x is true at position x if there is an a occurring at x or a later position. Now,

$$\psi(y) : \forall x \, [((x < y) \wedge \neg Q_c x) \Rightarrow (\exists y \, [(x \leq y) \wedge Q_a y])]$$

which has y as a free variable is true at position y if all positions x before y that do not hold a c satisfy the property $\phi(x)$. Finally,

$$\eta(x) : \exists y \, [(x < y) \wedge Q_d y \wedge \forall x \, [((x < y) \wedge \neg Q_c x) \Rightarrow (\exists y \, [(x \leq y) \wedge Q_a y])]]$$

checks that there is a $y > x$ holding d and satisfying $\psi(y)$. Putting it all together, we see that if x holds the last a then it satisfies $\eta(x)$ iff its suffix lies in $c^* d \{b, c, d\}^*$. Indeed, any

[6] A more formal discussion is given in [ST03]

y occurring after the last a satisfies $\psi(y)$ if and only if all positions between that last a and y hold a c.

We denote as $\mathbf{FO_k}[\mathcal{N}]$, $\mathbf{MOD}_k[\mathcal{N}]$ and $\mathbf{FO}+\mathbf{MOD}_k[\mathcal{N}]$ the different classes of first-order sentences constructed with at most k distinct variables.

5.1.1. *The First-Order Case.*

Kamp showed that a language is starfree if and only if it can be defined in **LTL** (linear temporal logic) [Kam68]. We formally describe this logic in the next subsection and show how an **LTL** formula can easily be translated into an equivalent $\mathbf{FO_3}[<]$ sentence. Thus

Theorem 5.3. $\mathrm{L}(\mathbf{FO}[<]) = \mathrm{L}(\mathbf{FO_3}[<]) = \mathrm{L}(\mathbf{LTL}) = \mathrm{L}(\mathbf{A}) = \mathcal{S}F$

Lemma 3.5 provides us with a characterization of the expressive power of $\mathbf{FO_1}[<]$. The case of two-variable sentences was first studied by Etessami, Vardi and Wilke who showed that a language is definable in $\mathbf{FO}_2[<]$ if and only if it can be defined in unary temporal logic, i.e. by an **LTL** sentence using only unary temporal operators [EVW02]. The problem of deciding whether a language was definable in this logic was later settled through the algebraic characterization of this class, given by Thérien and Wilke [TW98].

Let us quickly review the mechanics of our proofs in Section 4. We decompose sentences of quantifier depth $k+1$ as images of sentences of depth 1 under a substitution of formulas of quantifier depth k. Since by Lemma 4.1 any formula $\phi(x)$ of quantifier depth k can be written as boolean combinations of formulas of the form $Q_a x \wedge \rho_{[<x]} \wedge \chi_{[>x]}$ we can conclude that the pointed languages definable by such formulas are exactly the pointed languages in $\mathrm{P}(\mathbf{V_k})$ and this makes our inductive proof possible.

In the case of two-variable sentences, we cannot hope to find an analog of Lemma 4.1: if ρ is a sentence using only two variables x, y it is not possible to construct the relativization $\rho_{[<x]}$ without introducing new variables. To circumvent this problem we choose to decompose two-variable sentences of depth $k+1$ as the images of sentences of depth k under a substitution of formulas of quantifier depth 1.

When considering substitutions in the two-variable context, we need to worry about preserving the two-variable property. In other words, if Λ is a class of two-variable sentences and Γ is a class of two variable formulas with at most one free variable, we denote as $\Lambda \circ \Gamma$ the class of sentences which are boolean combinations of sentences in Γ and sentences obtained from a Λ sentence by replacing each occurrence of a predicate $Q_a x$ (resp. $Q_a y$) by a formula $\phi_a(x)$ of Λ (resp. $\phi_a(y)$). The block-product/substitution principle still holds true under this restricted notion of substitutions [TT05b].

While we analyzed $\mathbf{FO}[<]$ sentences by starting from the outermost quantifiers it is much more convenient to begin our study of a two-variable $\mathbf{FO}_2[<]$ sentence ϕ by looking at an innermost quantifier. Indeed, since ϕ uses only two variables, its most deeply nested subformula containing a quantifier is always of the form $\exists y\ \psi(x,y)$ or $\exists x\ \psi(x,y)$, where $\psi(x,y)$ is quantifier-free. We therefore isolate the $\mathbf{FOF}_1[<]$ subformulas of ϕ which are boolean combinations of formulas of the form $\exists y\ [(x * y) \wedge Q_a y]$ for $* \in \{<, >, =\}$ and formulas of this form with the roles of x and y reversed.

Let $\mathbf{Q_{2,k}}$ denote the class of $\mathbf{FO}_2[<]$ sentences of quantifier depth at most k. From the observations of the previous paragraph we have $\mathbf{Q_{2,k+1}} = \mathbf{Q_{2,k}} \circ \mathbf{FOF_1}[<]$ and one obtains

Lemma 5.4. *Let* $\mathbf{W_1} = \mathbf{SL}$ *and* $\mathbf{W_{i+1}} = \mathbf{W_i} \,\square\, \mathbf{SL}$ *for each* $i \geq 1$. *Then for each* $k \geq 1$ *we have* $\mathrm{L}(\mathbf{W_k}) = \mathrm{L}(\mathbf{Q_{2,k}})$.

Proof. The proof is a straightforward induction. The base case

$$\mathrm{L}(\mathbf{W_1}) = \mathrm{L}(\mathbf{SL}) = \mathrm{L}(\mathbf{FO}_1[<]) = \mathrm{L}(\mathbf{Q_{2,1}})$$

is given by Lemma 3.5.

For the induction step, assume $\mathrm{L}(\mathbf{W_k}) = \mathrm{L}(\mathbf{Q_{2,k}})$. We know by Lemma 3.5 that $\mathrm{P}(\mathbf{SL}) = \mathrm{P}(\mathbf{FOF}_1[<])$ and by the block-product/substitution principle

$$\mathrm{L}(\mathbf{Q_{2,k+1}}) = \mathrm{L}(\mathbf{Q_{2,k}} \circ \mathbf{FOF_1}[<]) = \mathrm{L}(\mathbf{W_k} \,\square\, \mathbf{SL}) = \mathrm{L}(\mathbf{W_{k+1}}). \qquad \square$$

Thus, a language L is definable by an $\mathbf{FO}_2[<]$ sentence if and only if its syntactic monoid M belongs to one of the pseudovarieties

$$\mathbf{W_k} = (\ldots((\underbrace{\mathbf{SL} \,\square\, \mathbf{SL}) \,\square\, \mathbf{SL}) \,\square \ldots \mathbf{SL}}_{k \text{ times}}).$$

Note that this iterated block product uses the weaker left-to-right bracketing. The union of the $\mathbf{W_k}$ is the smallest pseudovariety $\mathbf{W}$ satisfying $\mathbf{W} \,\square\, \mathbf{SL} = \mathbf{W}$. Let $\mathbf{DA}$ denote the pseudovariety of monoids satisfying $(xy)^\omega y (xy)^\omega = (xy)^\omega$.

Theorem 5.5 ([ST02])**.** *The pseudovariety* $\mathbf{DA}$ *is the smallest satisfying* $\mathbf{DA} \,\square\, \mathbf{SL} = \mathbf{DA}$.

Combining this result with Lemma 5.4 we get the following theorem of Thérien and Wilke [TW98]:

Corollary 5.6. $\mathrm{L}(\mathbf{FO}_2[<]) = \mathrm{L}(\mathbf{DA})$.

This immediately provides an algorithm for deciding if a regular language is definable by an $\mathbf{FO}_2[<]$ sentence because the pseudovariety $\mathbf{DA}$ is decidable. This pseudovariety admits a number of interesting characterizations [TT02] and, in particular, the regular languages whose syntactic monoids lie in $\mathbf{DA}$ have a nice combinatorial description. In fact, the original proof of Corollary 5.6 relied upon this characterization rather than on the decomposition of $\mathbf{DA}$ in terms of weakly iterated block products. For regular languages $L_0, \ldots, L_k \subseteq \Sigma^*$ and letters $a_1, \ldots, a_k \in \Sigma$, we say that the concatenation $L = L_0 a_1 L_1 \ldots a_k L_k$ is *unambiguous* if for each $w \in L$ there exists a unique factorization of w as $w = w_0 a_1 w_1 \ldots a_k w_k$ with $w_i \in L_i$.

Theorem 5.7 ([Sch76])**.** *A language* $L \subseteq \Sigma^*$ *has its syntactic monoid in* $\mathbf{DA}$ *if and only if* L *is the disjoint union of unambiguous concatenations of the form* $\Sigma_0^* a_1 \Sigma_1^* \ldots a_k \Sigma_k^*$*, where* $a_i \in \Sigma$ *and* $\Sigma_i \subseteq \Sigma$.

Furthermore, Pin and Weil show that L lies in $\mathrm{L}(\mathbf{DA})$ if and only if both L and its complement lie in the second level of the Straubing-Thérien hierarchy. Thus, L is definable in $\mathbf{FO}_2[<]$ if and only if it is definable in both $\Sigma_2[<]$ and in $\Pi_2[<]$.

Theorem 5.8 ([TW98])**.** $\mathbf{FO}_2[<] = \Sigma_2[<] \cap \Pi_2[<]$.

Example 5.9.
In Example 5.2, we gave two first-order sentences defining the language $L = \Sigma^* a c^* d \{b, c, d\}^*$, the second of which was $\mathbf{FO}_2[<]$. Note first that this concatenation is unambiguous: if a word w belongs to L then there is a unique factorization $w = w_0 a w_1 d w_2$ such that $w_1 \in c^*$ and $w_2 \in \{b, c, d\}^*$ because w_1 must start right after the last occurrence of the letter a in w and must end at the first occurrence of d after this a. Hence, the syntactic monoid of L

lies in **DA**. We can also define L using the following $\Sigma_2[<]$ sentence which simply reflects the structure of the regular expression for L:

$$\exists x \exists y \forall z \Big((x < y) \wedge Q_a x \wedge Q_d y \wedge [(x < z < y) \to Q_c z] \wedge [(z > y) \to (Q_b z \vee Q_c z \vee Q_d z)] \Big)$$

But the following $\Pi_2[<]$ sentence also defines L:

$$\forall x \forall y \forall z \exists s \exists t \exists u \Big(Q_a t \wedge Q_d u \wedge [((x < y) \wedge Q_a x \wedge Q_d y) \to ([(x < z < y) \to Q_c z] \vee ((x < s) \wedge Q_a s) \vee ((x < s < y) \wedge Q_d s))] \Big)$$

Indeed, this sentence relies on the fact that a word belongs to L if it contains at least one a, contains at least one d and is such that for any position x holding a and any later y holding d either all positions between x and y hold c or there exists an a occurring later than x or a d occurring between x and y.

Example 5.10.
We gave in Example 2.1 a $\Sigma_2[<]$ sentence defining the language $K = \{a, b, c\}^* ac^* a \{a, b, c\}^*$. Elementary computations can show that the syntactic monoid U of K consists of the six elements $\{1, a, b, ab, ba, 0\}$ with multiplication specified[7] by $aa = 0$, $bb = b$, $aba = a$, $bab = b$ and $0u = u0 = 0$ for all $u \in U$. In particular, ba is idempotent and if $x = b$ and $y = a$, we have

$$(ba)^\omega a (ba)^\omega = baaba = 0 \neq (ba)^\omega.$$

Thus U does not belong to **DA** and K cannot be defined by a $\Pi_2[<]$ sentence or by an $\mathbf{FO}_2[<]$ sentence.

5.1.2. *Two-Variable Sentences with Modular Quantifiers.*

To characterize the expressive power of $\mathbf{MOD}_2[<]$ and $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentences, we can precisely follow the proof paradigm used in the $\mathbf{FO}_2[<]$ case above. Since we have $\mathrm{L}(\mathbf{MOD}_1[<]) = \mathrm{L}(\mathbf{Ab})$ and $\mathrm{P}(\mathbf{MOD}_1[<]) = \mathrm{P}(\mathbf{Ab})$ we are naturally led to consider the smallest pseudovariety $\mathbf{V}$ such that $\mathbf{V} \,\square\, \mathbf{Ab} = \mathbf{V}$ (for the $\mathbf{MOD}_2[<]$ case) and the smallest pseudovariety $\mathbf{W}$ such that $\mathbf{W} \,\square\, \mathbf{Ab} = \mathbf{W}$ and $\mathbf{W} \,\square\, \mathbf{SL} = \mathbf{W}$ (for the $\mathbf{FO}^+\mathbf{MOD}_2[<]$ case).

Theorem 5.11 ([ST02])**.** *The pseudovariety* $\mathbf{G_{sol}}$ *is the smallest pseudovariety satisfying* $\mathbf{G_{sol}} \,\square\, \mathbf{Ab} = \mathbf{G_{sol}}$.

The pseudovariety $\mathbf{DA} \square \mathbf{G_{sol}}$ *is the smallest pseudovariety satisfying* $(\mathbf{DA} \square \mathbf{G_{sol}}) \square \mathbf{Ab} = \mathbf{DA} \,\square\, \mathbf{G_{sol}}$ *and* $(\mathbf{DA} \,\square\, \mathbf{G_{sol}}) \,\square\, \mathbf{SL} = \mathbf{DA} \,\square\, \mathbf{G_{sol}}$.

This theorem yields

Corollary 5.12 ([ST03])**.** *A language* L *is definable in* $\mathbf{FO}^+\mathbf{MOD}_2[<]$ *if and only if its syntactic monoid* $M(L)$ *lies in* $\mathbf{DA} \,\square\, \mathbf{G_{sol}}$ *and is furthermore definable in* $\mathbf{MOD}_2[<]$ *if* $M(L)$ *is a solvable group.*

[7] Note that despite the similarity, the monoid U is not isomorphic to the syntactic monoid B_2 of $(ab)^*$ because $bb = b$ in U and $bb = 0$ in B_2.

Let us denote as $\Sigma_2 \circ \mathbf{MODF}[<]$ the class of $\mathbf{FO}^+\mathbf{MOD}[<]$ sentences which, as the terminology suggests, are positive[8] boolean combinations of sentences obtained by applying to a $\Sigma_2[<]$ sentence a substitution using formulas containing only modular quantifiers. We define $\Pi_2 \circ \mathbf{MODF}[<]$ similarly. Straubing and Thérien obtained the following analog of Theorem 5.8:

Theorem 5.13 ([ST03]). $(\Sigma_2 \circ \mathbf{MODF}[<]) \cap (\Pi_2 \circ \mathbf{MODF}[<]) = \mathbf{FO}^+\mathbf{MOD}_2[<]$.

Although Corollary 5.12 gives an exact algebraic characterization of $\mathbf{FO}^+\mathbf{MOD}_2[<]$, it does not provide an effective way of testing if a given regular language is definable in this logic because the pseudovariety $\mathbf{DA} \,\square\, \mathbf{G_{sol}}$ is not known to be decidable. We have $\mathbf{DA} \,\square\, \mathbf{G_{sol}} \subseteq (\mathbf{DA} \,\square\, \mathbf{G}) \cap \mathbf{M_{sol}}$ and the latter two pseudovarieties are decidable but the containment is strict. Straubing and Thérien show that $\mathbf{DA} \,\square\, \mathbf{G_{sol}}$ is decidable if and only if the smaller pseudovariety $\mathbf{SL} \,\square\, \mathbf{G_{sol}}$ is decidable [ST03]. The latter question is an outstanding open problem in combinatorial group theory with deep implications [MSW01].

Example 5.14.
Let us once again consider the language $L = (ab)^*$. Recall that L's syntactic monoid is the six element monoid $B_2 = \{1, a, b, ab, ba, 0\}$ whose multiplication is specified by $aba = a$, $bab = b$, $aa = 0$, $bb = 0$ and $x0 = 0x = 0$ for all $x \in B_2$. We mentioned that B_2 is aperiodic and gave an $\mathbf{FO}[<]$ sentence defining L. However, in B_2 we have $(ab)^\omega b (ab)^\omega = 0 \neq (ab)^\omega$ and so $B_2 \notin \mathbf{DA}$. Hence, L is not definable in $\mathbf{FO}_2[<]$. On the other hand one can show that B_2 belongs to the pseudovariety $\mathbf{DA} \,\square\, \mathbf{G_{sol}}$. While we could argue for this fact in algebraic terms, it is sufficient to show that the language L can be defined by an $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence. The language $(ab)^*$ consists of words of even length with a on every odd position and b on every even position so it is defined by the two-variable sentence

$$(\exists^{0 \bmod 2} x \ \mathrm{T}) \wedge \forall x \, [(Q_a x \to \exists^{0 \bmod 2} y \ (y < x)) \wedge (Q_b x \to \exists^{1 \bmod 2} y \ (y < x))].$$

This sentence is in fact $\Pi_1 \circ \mathbf{MODF}_1[<]$.

In particular this example proves that $(\mathbf{DA} \,\square\, \mathbf{G_{sol}}) \cap \mathbf{A} \neq \mathbf{DA}$ and so, somewhat counter-intuitively, there are star-free languages, i.e. $\mathbf{FO}[<]$ definable languages, which are not definable in $\mathbf{FO}_2[<]$ but are definable in $\mathbf{FO}^+\mathbf{MOD}_2[<]$. On the other hand the syntactic monoid U presented in Example 5.10 is the smallest aperiodic monoid that does not lie in $\mathbf{DA} \,\square\, \mathbf{G_{sol}}$ and so $\Sigma^* a c^* a \Sigma^*$ is definable in $\mathbf{FO}[<]$ but not in $\mathbf{FO}^+\mathbf{MOD}_2[<]$.

One can extend Corollary 5.12 to show that the pointed languages definable by a $\mathbf{MODF}_2[<]$ formula are exactly the pointed languages recognized by solvable groups. This yields an interesting corollary: any two-variable $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence is equivalent to a two-variable sentence in which no existential or universal quantifier appears in the scope of a modular quantifier[9]. Indeed this class of sentences is just $\mathbf{FO}_2[<] \circ \mathbf{MODF}_2[<]$ and, once again, the block-product/substitution principle yields

$$\mathrm{L}(\mathbf{FO}_2[<] \circ \mathbf{MODF}_2[<]) = \mathrm{L}(\mathbf{DA} \,\square\, \mathbf{G_{sol}}) = \mathrm{L}(\mathbf{FO}^+\mathbf{MOD}_2[<]).$$

In fact, it is possible to provide explicit rules for rewriting a two-variable $\mathbf{FO}^+\mathbf{MOD}$ sentence so that all modular quantifiers are pushed within the scope of existential and universal

[8] Note that since the negation of a $\Sigma_2[<]$ sentence is not in general a $\Sigma_2[<]$ sentence, we must avoid negation in the definition of the class.

[9] Note that this is *not* true in the case of sentences with an unbounded number of variables.

quantifiers [ST03] but the detour through algebra avoids the technical complications of this construction.

It is natural to ask whether one can symmetrically rewrite any $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence such that no modular quantifier lies in the scope of an existential or universal quantifier. In other words, we would like to understand the expressive power of the class of sentences $\mathbf{MOD}_2[<] \circ \mathbf{FOF}_2[<]$. Unfortunately, we cannot directly use the block-product substitution principle because we do not have an algebraic characterization of the class of pointed languages $\mathrm{P}(\mathbf{FOF}_2[<])$. Rather, we choose to view this class of sentences as the union over all k of the classes

$$\mathbf{MOD}_2[<] \circ \underbrace{\mathbf{FOF}_1[<] \circ \ldots \circ \mathbf{FOF}_1[<]}_{k \text{ times}}.$$

Since $\mathrm{P}(\mathbf{FOF}_1[<]) = \mathrm{P}(\mathbf{SL})$ it follows that a language L is definable by an $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence in which no modular quantifier appears in the scope of an existential or universal quantifier if and only if the syntactic monoid $M(L)$ lies in one of the pseudovarieties

$$\mathbf{S_k} = (\ldots((\mathbf{G_{sol}} \,\square\, \underbrace{\mathbf{SL}) \,\square\, \mathbf{SL}) \ldots \mathbf{SL}) \,\square\, \mathbf{SL}}_{k \text{ times}};$$

and is furthermore definable by an $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence in which no modular quantifier appears in the scope of any other quantifier if and only if $M(L)$ lies in one of the

$$\mathbf{T_k} = (\ldots((\mathbf{Ab} \,\square\, \underbrace{\mathbf{SL}) \,\square\, \mathbf{SL}) \ldots \mathbf{SL}) \,\square\, \mathbf{SL}}_{k \text{ times}}.$$

It is possible to show that for any k the pseudovarieties $\mathbf{S_k}$, $\mathbf{T_k}$ are decidable using the notion of kernels of monoid morphisms [Til87] (see also [TW04] for an application to logic). In any case, we are once again more interested in deciding membership in the union of the $\mathbf{S_k}$ or the $\mathbf{T_k}$. Let $\mathbf{DO}$ be the pseudovariety of finite monoids satisfying the identity $(xy)^\omega(yx)^\omega(xy)^\omega = (xy)^\omega$.

Lemma 5.15 ([TT05b])**.** *Let* $\mathbf{DO} \cap \mathbf{M_{sol}}$ *and* $\mathbf{DO} \cap \overline{\mathbf{Ab}}$ *denote the pseudovarieties consisting of monoids in* $\mathbf{DO}$ *whose subgroups are respectively solvable and Abelian. Then* $\bigcup_k \mathbf{S_k} = \mathbf{DO} \cap \mathbf{M_{sol}}$ *and* $\bigcup_k \mathbf{T_k} = \mathbf{DO} \cap \overline{\mathbf{Ab}}$*.*

This immediately yields

Corollary 5.16. *A language L is definable by an $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence in which no modular quantifier appears in the scope of an existential or universal quantifier if and only if its syntactic monoid M lies in $\mathbf{DO} \cap \mathbf{M_{sol}}$ and definable by a sentence in which no modular quantifier appears in the scope of another quantifier if and only M lies in $\mathbf{DO} \cap \overline{\mathbf{Ab}}$.*

Example 5.17.
Let us return to Example 2.4. It can be explicitly shown that the syntactic monoid $M(K)$ of $K = (b^*ab^*a)^*b\Sigma^*$ lies in $\mathbf{DA} \square \mathbf{G_{sol}}$ and, correspondingly, there exists an $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence defining K:

$$\exists x\, [Q_b x \wedge \exists^{0 \bmod 2} y\, [(y < x) \wedge Q_a y]].$$

The modular quantifier lies in the scope of the existential quantifier and we want to show that it cannot be pulled out. Indeed, by simple calculation one can see that $M(K)$ contains elements $\{1, a, b, ab, ba, aba, 0\}$ with multiplication given by $aa = 1$, $bb = b$, $bab = b$, $abab = 0$

and $0s = s0 = 0$ for all s. In particular aba and $b = baa$ are idempotents. Choosing $u = a$ and $v = ba$, we have

$$(uv)^\omega(vu)^\omega(uv)^\omega = (aba)^\omega(baa)^\omega(aba)^\omega = abababa = 0 \neq (aba)^\omega = (uv)^\omega$$

so $M(K)$ violates the identity defining **DO** and K cannot be defined by an $\mathbf{FO}+\mathbf{MOD_2}[<]$ sentence in which the modular quantifiers lie outside the scope of the ordinary quantifiers.

The same type of argument also shows that $(ab)^*$ cannot be defined by an $\mathbf{FO}+\mathbf{MOD_2}[<]$ sentence in which the modular quantifiers lie outside the scope of the ordinary quantifiers.

Example 5.18.
Consider for contrast the language of Example 2.5: we noted at the start of this section that the language L of words over $\{a, b, c\}^*$ such that the position holding the first a has a suffix containing an even number of c's is definable by the $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence

$$\exists x \forall y\, [Q_a x \wedge ((y < x) \Rightarrow \neg Q_a y) \wedge \exists^{0 \bmod 2} y\, ((x < y) \wedge Q_c y)].$$

One can verify that the syntactic monoid of L lies in $\mathbf{DO} \cap \overline{\mathbf{Ab}}$. In the above sentence, the modular quantifier appears within the scope of the leading existential quantifier but can in fact be pulled out: the sentence

$$\exists^{0 \bmod 2} x\ [Q_c x \wedge (\exists y\, ((y < x) \wedge Q_a y \wedge (\forall x\, [(y < x) \Rightarrow \neg Q_a x])]$$

asserts that there are an even number of c's which appear after the first occurrence of a and thus also defines L.

To conclude our discussion on two-variable sentences, note that although the successor relation is definable in $\mathbf{FO}[<]$ it is not possible in general to transform an $\mathbf{FO}^+\mathbf{MOD}_2[<, S]$ sentence into an equivalent $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence. A precise characterization of the class $\mathbf{FO}_2[<, S]$ in terms of syntactic semigroups is nonetheless given in [TW98].

5.2. **Temporal Logic.**

The idea of using weakly-iterated block-products to characterize the expressive power of two-variable $\mathbf{FO}^+\mathbf{MOD}[<]$ sentences came originally from the study of temporal logics. Such logics are widely used in hardware and software verification because they are able to express properties of dynamic processes in a natural and intuitive way.

A linear temporal logic formula (**LTL**) over the alphabet Σ is built from atomic formulas which are either one of the boolean constants T and F or one of the letters in Σ. We want to think of a word w satisfying the formula a at 'time' i if the ith letter of w is an a. More complex formulas are constructed from these atomic ones using boolean connectives and a certain set of temporal operators. We focus here on the cases where these operators are the unary operators $\diamondsuit$ (eventually in the future) and $\diamondsuit$ (eventually in the past) or the binary operators U (until) and S (since). The terminology of course stresses the intended meaning of these operators and we can formally define the semantics of an **LTL** formula ϕ over Σ for pointed words (w, p) with $w \in \Sigma^*$ as follows.

- For any (w, i) we have $(w, i) \models \mathrm{T}$ and $(w, i) \not\models \mathrm{F}$;
- For $a \in \Sigma$ we have $(w, i) \models a$ if and only if $w_i = a$;
- $(w, i) \models \diamondsuit\phi$ if there exists $i < j \leq |w|$ such that $(w, j) \models \phi$;
- $(w, i) \models \diamondsuit\phi$ if there exists $1 \leq j < i$ such that $(w, j) \models \phi$;

- $(w,i) \models \phi U \psi$ if there exists $i < j \leq |w|$ such that $(w,j) \models \psi$ and $(w,i') \models \phi$ for all $i < i' < j$;
- $(w,i) \models \phi S \psi$ if there exists $1 \leq j < i$ such that $(w,j) \models \psi$ and $(w,i') \models \phi$ for all $j < i' < i$;

Note that the **LTL** sentences $\diamondplus\phi$ and $\top \mathrm{U}\, \phi$ are equivalent and so the Until and Since operators are sufficient to obtain the full expressive power of **LTL**. Any **LTL** formula ϕ naturally defines a pointed language $\mathrm{P}_\phi = \{(w,i) : (w,i) \models \phi\}$. We also associate to ϕ the language $\mathrm{L}_\phi = \{w : (w,0) \models \phi\}$. If Φ is a class of **LTL** formulas, we similarly denote by $\mathrm{L}(\Phi)$ and $\mathrm{P}(\Phi)$ respectively the classes languages and pointed languages defined by a formula of Φ.

As we mentioned earlier, Kamp [Kam68] showed that $\mathrm{L}(\mathbf{LTL}) = \mathrm{L}(\mathbf{FO}[<])$ and in fact $\mathrm{P}(\mathbf{LTL}) = \mathrm{P}(\mathbf{FOF}[<])$. The containment from left to right is rather easy to obtain by induction on the structure of the **LTL** formulas. The atomic **LTL** formula a defines the set of pointed words (w,i) having the letter a in position i and thus corresponds to the formula $Q_a x$. Suppose by induction that for the **LTL** formulas ϕ and ψ we can construct $\mathbf{FOF}[<]$ formulas $\tau(x), \rho(x)$ such that $\mathrm{P}(\phi) = \mathrm{P}(\tau(x))$ and $\mathrm{P}(\psi) = \mathrm{P}(\rho(x))$ then the **LTL** formula $\phi U \psi$ defines the same pointed language as

$$\eta(x) : \exists y \forall z\ \rho(y) \wedge ((x < y < z) \Rightarrow \tau(z)).$$

The translation of the other three temporal operators can be obtained similarly. Note also that the structure of $\eta(x)$ allows us to construct this formula using only three variables and so $\mathrm{L}(\mathbf{LTL}) \subseteq \mathrm{L}(\mathbf{FO}_3)$. The inclusion $\mathrm{L}(\mathbf{FO}[<]) \subseteq \mathrm{L}(\mathbf{LTL})$ essentially amounts to showing $\mathrm{L}(\mathbf{FO}[<]) \subseteq \mathrm{L}(\mathbf{FO}_3[<])$ [Kam68, IK89].

For two classes Λ, Γ of **LTL** formulas we denote as $\Lambda \circ \Gamma$ the class of **LTL** formulas which are boolean combinations of formulas in Γ and formulas obtained from a Λ formula by replacing each occurrence of the atomic formula a by a formula $\phi_a \in \Gamma$. The block-product/substitution principle carries over to temporal logic: if there are pseudovarieties $\mathbf{V}, \mathbf{W}$ such that $\mathrm{L}(\Lambda) = \mathrm{L}(\mathbf{V})$, $\mathrm{P}(\Gamma) = \mathrm{P}(\mathbf{W})$ and $\mathrm{L}(\Gamma) \subseteq \mathrm{L}(\mathbf{V} \,\square\, \mathbf{W})$ then $\mathrm{L}(\Lambda \circ \Gamma) = \mathrm{L}(\mathbf{V} \,\square\, \mathbf{W})$.

The class of *unary temporal logic* formulas **UTL** is the subclass of **LTL** consisting of formulas constructed without the binary operators U, S. There is a natural hierarchy $\mathbf{UTL}_1 \subseteq \mathbf{UTL}_2 \subseteq \ldots$ within **UTL** defined by the nesting depth of the $\diamondplus, \diamondminus$ operators. We clearly have

$$\mathbf{UTL}_k = \underbrace{\mathbf{UTL}_1 \circ \ldots \circ \mathbf{UTL}_1}_{k \text{ times}}.$$

Lemma 5.19. $\mathrm{L}(\mathbf{UTL}_1) = \mathrm{L}(\mathbf{SL})$ *and* $\mathrm{P}(\mathbf{UTL}_1) = \mathrm{P}(\mathbf{SL})$.

Proof sketch. Any $\mathbf{UTL}_1$ formula is a boolean combination of formulas of the form a, $\diamondplus a$ or $\diamondminus a$. The rest of the argument is similar to the proof of part 1 of Lemma 3.5. □

Thus, the block-product/substitution principle insures:

Corollary 5.20 ([EVW02, TW02, ST02])**.**
For each k, $\mathrm{L}(\mathbf{UTL}_k) = \mathrm{L}((\ldots(\mathbf{SL} \,\square\, \mathbf{SL}) \,\square \ldots) \,\square\, \mathbf{SL}))$. *Moreover,*

$$\mathrm{L}(\mathbf{UTL}) = \mathrm{L}(\mathbf{DA}) = \mathrm{L}(\mathbf{FO_2}[<]) = \mathrm{L}(\mathbf{\Sigma_2}[<]) \cap \mathrm{L}(\mathbf{\Pi_2}[<]).$$

Example 5.21. We argued in Example 5.9 that the syntactic monoid of the language $L = \Sigma^* a c^* d \{b,c,d\}^*$ lies in **DA** and exhibited $\Sigma_2[<]$ and $\Pi_2[<]$ sentences defining L (an

equivalent $\mathbf{FO}_2[<]$ sentence was also given in Example 5.2). In temporal terms, L can be described as the set of words which contain an a that has no other a in its future but has in its future an occurrence of d with the property that each b or d in the past of this occurrence of d contains an a in its future.

$$\Diamond\!\!\!\!+ \left[a \wedge (\neg \Diamond\!\!\!\!+ a) \wedge (\Diamond\!\!\!\!+ (d \wedge [\neg \Diamond\!\!\!\!- ((b \vee d) \wedge \neg \Diamond\!\!\!\!+ a)]))\right].$$

By contrast $K = \{a,b,c\}^* ac^* a \{a,b,c\}^*$ has a syntactic monoid which is aperiodic but outside of **DA** (Example 5.10) and so K is definable in **LTL** but not in **UTL**.

The *Until/Since hierarchy* $\{\mathbf{USH}_k\}_{k\geq 0}$ within **LTL** corresponds to the nesting depth of the Until and Since operators (the unary operators do not contribute to the depth of a formula). We set $\mathbf{LTL}_0 = \mathbf{UTL}$. We have

$$\mathbf{USH}_k = \mathbf{UTL} \circ \underbrace{\mathbf{USH}_1 \circ \ldots \circ \mathbf{USH}_1}_{k \text{ times}}.$$

The Until/Since hierarchy was introduced by Etessami and Wilke [EW00] who proved that the hierarchy was infinite. The algebraic characterization of the levels of the Until/Since hierarchy was given by Thérien and Wilke [TW04]:

Theorem 5.22. *Let* **RB** *be the pseudovariety of monoids satisfying* $x^2 = x$ *and* $xyxzx = xyzx$. *Then*

$$\mathrm{L}(\mathbf{USH}_k) = L(((\mathbf{DA} \,\square\, \underbrace{\mathbf{RB}) \,\square\, \mathbf{RB}) \ldots) \,\square\, \mathbf{RB}}_{k \text{ times}}).$$

Roughly speaking, the proof links pointed languages of $\mathbf{USH}_1$ and pointed languages of $\mathrm{P}(\mathbf{RB})$. However a number of technical hurdles have to be overcome. This theorem also guarantees that the levels of the Until/Since hierarchy are decidable although the complexity of the algorithms provided in [TW04] is prohibitive.

The two temporal operators next and previous are also often used in the construction of **LTL** sentences. The additional expressive power offered by these operators is closely linked to the extra power afforded by the successor numerical predicate in first-order sentences and, at least intuitively, this is not a major surprise. Standard methods allow algebraic characterizations of the expressive power of the various levels of the Until/Since hierarchy and of **UTL** when next and previous operators are available [TW98, TW04].

In the context of software and hardware verification, 'future' operators $U, \Diamond\!\!\!\!+, next$ are more suited to express properties and the 'past' operators $\Diamond\!\!\!\!-$ and S are not so standard in **LTL**. Kamp in fact shows that the until operator U is sufficient to obtain the full expressive power of $\mathbf{LTL} = \mathbf{FO}[<]$. When future operators are the only ones available substitutions only allow additional information on the suffix of a given position and so the two-sided nature of the block-product makes it unsuited for the analysis. However, one can instead consider reverse semidirect products and obtain the correct analog of the principle. Cohen, Pin and Perrin used this idea to characterize the expressive power of unary future temporal logic [CPP93] and Thérien and Wilke later extended the idea to characterize the levels of the Until hierarchy [TW02]. Baziramwabo, McKenzie and Thérien also considered the extension of **LTL** in which new modular counting temporal operators are introduced [BMT99]. An early survey of Wilke provides an overview of the semigroup theoretic approach in the analysis of temporal logics [Wil01].

6. Logic, Algebra and Circuit Complexity

6.1. **Boolean Circuits.**

We have so far considered only the case of first-order formulas with order ($<$) as the sole numerical predicate. When **FO**+**MOD** sentences have access to non-regular predicates, their expressive power is dramatically increased and they can provide logical characterizations for a number of well-known classes of boolean circuit complexity.

A boolean circuit C on n boolean variables $w_1, \ldots, w_n$ is a directed acyclic graph with a distinguished output node of outdegree 0. A node of in-degree 0 is called an input node (or input gate) and is either labeled by one of the boolean constants $0, 1$ or by some boolean literal w_i or $\overline{w_i}$. Any other node g of C (including the output node) is labeled by some symmetric boolean function f_g chosen from some predetermined base. The most standard case has each inner node labeled either by the Or or the And function but we also consider the case where gates are labeled by the boolean function Mod_m which is 1 if the sum of its inputs is divisible by m and is 0 otherwise. Any gate g of a boolean circuit on n variables naturally computes a boolean function $v_g : \{0,1\}^n \to \{0,1\}$. If the gate g is an input node labeled by w_i (resp. $\overline{w_i}$) then $v_g(w) = 1$ if and only if $w_i = 1$ (resp. $w_i = 0$). If g is an inner node then a gate g' is an *input to* g if there is a directed edge (g', g) in the graph C. Naturally, if $g_1, \ldots, g_k$ are the inputs of g we set

$$v_g(w) = f_g(v_{g_1}(w), \ldots, v_{g_k}(w)).$$

If *out* is the output node of C then the *function computed by the circuit* is $C(w) = v_{out}(w)$. The *language accepted by the circuit* is the set $\{w \in \{0,1\}^n : C(w) = 1\}$ of n-bit strings on which the circuit outputs 1.

The *depth* d of a circuit C is the length of the longest path from an input node to the output node. The *size* s of C is the number of gates in C. We are also interested in considering circuits in which inputs w_i are not booleans but rather take values in some finite alphabet Σ. This can be handled either by using a binary encoding of Σ or by labeling input nodes by functions $w_i = a$ for some $a \in \Sigma$. The rest of our discussion is unaffected by these implementation details.

By definition a boolean circuit can only process inputs of some fixed length n but we are interested in using circuits as computing devices recognizing languages in Σ^*. This can be done by providing an infinite family $\mathcal{C}$ of circuits $\mathcal{C} = \{C_n\}_{n \geq 0}$ where the circuit C_i processes inputs of length i. In this case, we define the size $s(\mathcal{C})$ and the depth $d(\mathcal{C})$ of a circuit family as functions of the input size.

Note that for any subset $K \subseteq \mathbb{N}$, the language $\{w : |w| = k \wedge k \in K\}$ can be recognized by a family of circuits of depth 0 and size 1 since inputs of a given length are either all accepted or all rejected. If we do not impose any constraints on the constructibility of circuit families, boolean circuits are thus able to recognize undecidable languages. *Uniformity restrictions* on circuit families impose the existence of an (efficient) algorithm that computes some representation of the nth circuit C_n of a family. We say that a family of circuits $\mathcal{C}$ is *uniform* if such an algorithm exists and furthermore say that $\mathcal{C}$ is P-*uniform* (resp. L-*uniform*) if there is a polynomial time (resp. logarithmic space) algorithm which on input 1^n constructs C_n. An even more stringent requirement is that of dlogtime-uniformity which requires the existence of an algorithm which on input (n, i, j) computes in time $O(\log |n|)$ the type of the ith and jth gates of C_n and determines whether these gates are connected by a wire [BIS90].

We define some classical circuit complexity classes:

Definition 6.1. The boolean circuit complexity class non-uniform AC^0 is the class of languages which are computable by a family $\mathcal{C} = \{C_n\}_{n\geq 0}$ of circuits constructed with AND and OR gates with depth $d(\mathcal{C}) = O(1)$ and size $s(\mathcal{C}) = O(n^k)$ for some k.

Similarly, non-uniform CC^0 is the class of languages computable by families of circuits of bounded depth and polynomial size and constructed with gates MOD_m for some $m \geq 2$. Non-uniform ACC^0 is the class of languages computable by families of circuits of bounded depth and polynomial size and constructed with gates AND, OR and MOD_m for some $m \geq 2$.

Finally, non-uniform NC^1 is the class of languages computable by families of circuits of depth $O(\log n)$, polynomial-size and constructed with AND and OR gates of fan-in 2.

There are natural uniform versions of the above classes. By definition both AC^0 and CC^0 are subclasses of $\mathrm{ACC}^0 \subseteq \mathrm{NC}^1$. Moreover, L-uniform-$\mathrm{NC}^1$ is a subclass of L (logspace). The containment $\mathrm{AC}^0 \subseteq \mathrm{ACC}^0$ is known to be strict because the parity function (i.e. the MOD_2 function) cannot be computed by bounded depth AND,OR circuits of subexponential size [Ajt83, FSS84, Smo86]. It is conjectured that CC^0 is also strictly contained in ACC^0 and, in particular, that the AND function requires bounded depth MOD_m circuits of super-polynomial size. Despite an impressive body of work in circuit complexity [All97, Vol99], no such lower bound is known and even much weaker statements such as DLOGTIME-uniform $\mathrm{CC}^0 \neq \mathrm{NP}$ still elude proof.

These circuit classes have nice logical descriptions which were made explicit by Gurevich, Lewis, Barrington, Immerman and Straubing [GL84, Imm87, BIS90, Str94].

Theorem 6.2.
$\mathrm{AC}^0 = \mathbf{FO}$ *(i.e.* $\mathbf{FO}$ *extended with all numerical predicates) and* DLOGTIME-$\mathrm{AC}^0 = \mathbf{FO}[+, *]$.
$\mathrm{CC}^0 = \mathbf{MOD}$ *and* DLOGTIME-$\mathrm{CC}^0 = \mathbf{MOD}[+, *]$.
$\mathrm{ACC}^0 = \mathbf{FO}{+}\mathbf{MOD}$ *and* DLOGTIME-$\mathrm{ACC}^0 = \mathbf{FO}{+}\mathbf{MOD}[+, *]$.

Proof sketch. The statements about DLOGTIME uniformity are too technical to present succinctly [BIS90] but it is rather straightforward to prove, for instance, that $\mathrm{AC}^0 = \mathbf{FO}$. For the right to left containment, we need to build for any $\mathbf{FO}$ sentence ϕ a non-uniform AC^0 circuit family $\mathcal{C}$ that accepts exactly L_ϕ. We assume without loss of generality that ϕ is in prenex normal form:

$$\phi : \mathcal{Q}_1 x_1 \mathcal{Q}_2 x_2 \ldots \mathcal{Q}_k x_k \ \psi(x_1, \ldots, x_k)$$

where ψ is quantifier free and each $\mathcal{Q}_i$ is $\exists$ or $\forall$. Circuit C_n is obtained by using OR and AND gates to respectively represent the existential and universal quantifiers. Each of those gates has fan-in n so that a wire into the gate representing $\mathcal{Q}_i x_i$ represents one of the n possible values of x_i. Finally, for any choice of values $(x_1, \ldots, x_k)$ we need to build subcircuits computing the value of $\psi(x_1, \ldots, x_k)$: the atomic formulas of the form $Q_a x_i$ are evaluated using a query to the input variable x_i and the value of a numerical predicate $R(x_{i_1}, \ldots, x_{i_t})$ can be hardwired into the nth circuit since the value of R only depends on the value of the x_{i_j} and the input length n. Note that the size of the nth circuit built in this way is at most $c \cdot n^{k+1}$ for some $c \geq 1$.

To show $\mathrm{AC}^0 \subseteq \mathbf{FO}$, we first normalize our circuit family $\mathcal{C}$ so that each C_n of $\mathcal{C}$ is a tree of depth k which is leveled so that gates at level i in any circuit of the family are either all OR or all AND gates. Moreover, we insure that every non-input gate has fan-in exactly n so that we can think of these n wires as being indexed by positions in the input.

By extension, any sequence of k input positions can be viewed as a path from the output gate back to some input gate.

It is a simple exercise to show that the normalization process of our circuit family can be done so that the resulting family still has bounded depth and polynomial-size. The construction of an **FO** sentence defining the language accepted by $\mathcal{C}$ then follows naturally: if the family of circuits has depth k, the sentence has k quantifiers where existential and universal quantifiers are used to respectively represent levels of And gates and Or gates. We complete the construction by using a $k+1$-ary numerical predicate $R(i, x_1, \ldots, x_k)$ which is true if the path $(x_1, \ldots, x_k)$ from the output gate back to the input queries the ith bit of the input. Note also that the non-uniformity of the family of circuits can be handled easily since we allow the value of the numerical predicates to depend on the length of the input word. □

Note that the polynomial-size restriction in the definition of AC^0, CC^0 and ACC^0 is in some sense built into this correspondence with first-order logic. Lautemann [KLPT06] further noted that when arbitrary numerical predicates are used, the restriction of **FO**, **FO**+**MOD** and **MOD** to two variables correspond to a linear-size restriction on the corresponding circuits.

Theorem 6.3. *A language L is computable by a family of AC^0 (resp. CC^0, ACC^0) circuits of size $O(n)$ if and only if L is definable by a two-variable $\mathbf{FO}_2$ (resp. $\mathbf{MOD}_2$, $\mathbf{FO}$+$\mathbf{MOD}_2$) sentence with arbitrary numerical predicates.*

AC^0, CC^0 and ACC^0 (as well as a number of other important circuit complexity classes) also admit very interesting algebraic characterizations using the *programs over finite monoids* formalism. The idea first appeared in Chandra, Stockmeyer and Vishkin [CSV84] but was formalized and further developed by Barrington and Thérien [Bar89, BT88]. A number of lower bounds for restricted classes of circuits can be obtained through this approach [BST90, BS94, ST06]. A detailed account of this line of work is beyond our scope but we refer the interested reader to Straubing's book [Str94] or one of the surveys [MPT91, Str00, TT04, TT06].

6.2. Bounding the Expressive Power of FO+MOD: Partial Results.

The logical description of circuit classes suggests a natural incremental approach to obtaining strong complexity separation results such as the strict containment of non-uniform CC^0 in ACC^0 or of non-uniform-ACC^0 in Logspace. Such results amount to bounding the expressive power of **FO**+**MOD** or **MOD** and while this seems a deep mathematical challenge we can hope that for sufficiently simple classes of numerical predicates $\mathcal{N}$ it is at least possible to bound the expressive power of $\mathbf{FO}$+$\mathbf{MOD}[\mathcal{N}]$ and $\mathbf{MOD}[\mathcal{N}]$. On one hand $\mathbf{FO}$+$\mathbf{MOD}[Reg] = \mathbf{FO}$+$\mathbf{MOD}[<]$ contains only the regular languages with solvable monoids but, on the other hand, even bounding the expressive power of $\mathbf{FO}$+$\mathbf{MOD}[+, *]$ is beyond the capabilities of current lower bound technology and so it makes sense to consider classes of numerical predicates with intermediate expressive power.

The obvious target is of course $\mathcal{N} = \{+\}$. Lynch proved that Parity is not expressible in $\mathbf{FO}[+]$ [Lyn82a, Lyn82b] (see also [BIL$^+$01]). Building on work exposed in Libkin's book [Lib04], Roy and Straubing further showed that if p is a prime that does not divide q then the language Mod_p is not expressible in $\mathbf{FO}$+$\mathbf{MOD}_q[+]$ where the q subscript indicates that only quantifiers counting modulo q are used [RS06]. In later work, Behle

and Lange [BL06] translated the restriction of $\mathcal{N}$ to $\{+\}$ into a uniformity restriction on circuits. Lautemann et al. [LMSV01], Schweikardt [Sch05] and Lange [Lan04] all provided further evidence of the fairly weak expressive power of addition even in the case where **FO** is augmented by so-called counting quantifiers or majority quantifiers.

In a somewhat different direction, Nurmonen [Nur00] and Niwiński and Stolboushkin [SN97] considered logics equipped with numerical predicates of the form $y = kx$ for some integer k and in particular establish that there is no $\mathbf{FO}{+}\mathbf{MOD}_q[<, \{y = qx\}]$ sentence that defines the set of words whose length is divisible by p where p does not divide q.

If we are trying to exhibit a language L who cannot be defined in $\mathbf{FO}{+}\mathbf{MOD}[\mathcal{N}]$ for some class $\mathcal{N}$ of numerical predicates, it makes sense to choose L so that the predicates in $\mathcal{N}$ seem particularly impotent in a sentence defining L. This intuition is of course difficult to formalize but it led to the study of languages with a neutral letter. A letter $e \in \Sigma$ is said to be *neutral* for $L \subseteq \Sigma^*$ if for all $u, v \in \Sigma^*$ it holds that $uev \in L \Leftrightarrow uv \in L$. In other words e is neutral for L if e is equivalent to the empty word ϵ under the syntactic congruence of L. At least intuitively, it is difficult to construct circuits to recognize languages having a neutral letter because they cannot rely on the precise location of the relevant (i.e. non-neutral) letters of their input. By the same token, access to arbitrary numerical predicates seems of little help to define these languages. Lautemann and Thérien conjectured that every language with a neutral letter recognized in AC^0 is in fact a star-free regular language. The so-called Crane-Beach conjecture, was in fact refuted in [BIL$^+$01]: if $\mathcal{L}_e$ denotes the class of languages with a neutral letter, then there is a language in $(\mathbf{FO}[+, *] \cap \mathcal{L}_e) - \mathbf{FO}[<]$. Nevertheless, the same authors proved

$$\mathbf{FO}[+] \cap \mathcal{L}_e = \mathbf{FO}[<] \cap \mathcal{L}_e$$

and

$$BC(\Sigma_1) \cap \mathcal{L}_e = BC(\Sigma_1[<]) \cap \mathcal{L}_e$$

where BC denotes the boolean closure. Let $\mathbf{MOD}_p$ be the class of languages definable by a **MOD** sentence using only quantifiers that count modulo p for some prime p and arbitrary numerical predicates. Lautemann and the two current authors have shown [LTT06] that

$$\mathcal{L}_e \cap \mathbf{MOD}_p = \mathbf{MOD}_p[<] \cap \mathcal{L}_e.$$

The neutral letter hypothesis has shown useful in other similar contexts, in particular to obtain superlinear lower bounds for bounded-width branching programs [BS95] and in communication complexity [RTT98, TT05a, CKK$^+$07].

6.3. **The Circuit Complexity of Regular Languages.**

Regular languages are a fascinating case study in circuit complexity [BCST92, CS01, Pél92, PST97, Str94, TT06]. As we mentioned earlier, one of the most celebrated results in complexity theory is the lower bound on the size of AC^0 circuits computing the regular language PARITY. Moreover, from the results of [BT88, MPT91] the main current conjectures on separations of circuit complexity classes amount to answering questions about the circuit complexity of specific regular languages. For instance, CC^0 is strictly contained in ACC^0 if and only if AND is not in CC^0 and ACC^0 is strictly contained in the circuit class NC^1 if and only if regular languages with non-solvable syntactic monoids are not recognizable in ACC^0.

Some of these questions can be recast in purely model-theoretic terms [BCST92, Str92, Str94, STT93, Pél92]. Intuitively, the only numerical predicates that can be of any significant use in defining regular languages are the regular predicates described at the end of Section 4. For example, [BCST92] used the fact that the MOD_p-functions do not lie in AC^0 to show that a regular language is definable in **FO** iff it is definable in $\mathbf{FO}[Reg]$. If $\mathcal{R}$ denotes the class of regular languages then the conjectured separation of ACC^0 from NC^1 is equivalent to the statement

$$\mathcal{R} \cap \mathbf{FO}{+}\mathbf{MOD} = \mathbf{FO}{+}\mathbf{MOD}[Reg].$$

and, similarly, $\text{CC}^0 \neq \text{ACC}^0$ is equivalent to

$$\mathcal{R} \cap \mathbf{MOD} = \mathbf{MOD}[Reg].$$

These equivalences are discussed in full detail in [Str94] and we simply sketch here the argument for the first of them. Assume that $\text{ACC}^0 = \text{NC}^1$: since every regular language is in NC^1, we have $\mathcal{R} \cap \mathbf{FO}{+}\mathbf{MOD} = \mathcal{R} \cap \text{ACC}^0 = \mathcal{R} \cap \text{NC}^1 = \mathcal{R}$ whereas $\mathbf{FO}{+}\mathbf{MOD}[Reg] = \mathbf{FO}{+}\mathbf{MOD}[<]$ contains only those regular languages whose syntactic monoid is solvable (Theorem 4.4).

On the other hand, Barrington and Thérien [BT88] showed that any regular language whose syntactic monoid is not solvable (and therefore not definable in $\mathbf{FO}{+}\mathbf{MOD}[Reg]$, is complete for NC^1 under very simple reductions known as *non-uniform projections* or *programs*. Therefore, if $\text{ACC}^0 \neq \text{NC}^1$ then none of these languages lies in ACC^0 and $\mathcal{R} \cap \mathbf{FO}{+}\mathbf{MOD} = \mathbf{FO}{+}\mathbf{MOD}[<]$.

We can refine our questions about the circuit complexity of regular languages and ask how small the AC^0, CC^0 and ACC^0 circuits recognizing them can be. For AC^0, a surprising partial answer was provided by Chandra, Fortune and Lipton [CFL85] who show that any regular language computed by an AC^0 circuit can in fact be computed by an AC^0-circuit with only $O(ng^{-1}(n))$ wires (and thus gates) for any primitive recursive function g. The result in fact extends to ACC^0. The only regular languages known to be (and believed to be) in CC^0 are those definable in $\mathbf{MOD}_2[Reg]$ and, by Theorem 6.3, these can all be recognized with circuits with $O(n)$ gates. It is tempting to further conjecture that any regular language which is not definable in $\mathbf{FO}_2[Reg]$ (resp. $\mathbf{FO}{+}\mathbf{MOD}_2[Reg]$) is in fact not definable in $\mathbf{FO}_2$ (resp. $\mathbf{FO}{+}\mathbf{MOD}_2$) and therefore requires superlinear-size AC^0 (resp. ACC^0) circuits. In other words, superlinear-size lower bounds for AC^0 and ACC^0 circuits can conceivably be obtained through logical methods such as Ehrenfeucht-Fraïssé games showing that a given language is not $\mathbf{FO}_2$ or $\mathbf{FO}{+}\mathbf{MOD}_2$ definable.

Kouck, Pudlk and Thérien considered the class of regular languages (with a neutral letter) which are recognizable by ACC^0 circuits with only $O(n)$ *wires*.

Theorem 6.4. *If L is a regular language with a neutral letter then L can be recognized by a family of ACC^0 circuits with $O(n)$ wires if and only if $L \in \text{L}(\mathbf{DO} \cap \overline{\mathbf{Ab}})$ if and only if L is definable by an $\mathbf{FO}{+}\mathbf{MOD}_2[<]$ in which no modular quantifier lies in the scope of another quantifier.*

The superlinear lower bound needed to obtain this theorem requires significant work and relies on an extension of deep combinatorial results of Pudlk on superconcentrators [Pud94] and on a linear lower bound [TT05a] on the communication complexity of regular languages which do not belong to $\text{L}(\mathbf{DO} \cap \overline{\mathbf{Ab}})$.

The upper bound is based on a result of Bilardi and Preparata [BP90] which exhibits an AC^0 circuit with n inputs $x_1, \ldots, x_n$, $2n$ input gates and only $O(n)$ wires which on

input $\{0,1\}^n$ computes the OR function of each prefix $x_1 \dots x_i$ and each suffix $x_{i+1} \dots x_n$ of the input. To build circuits with $O(n)$ wires recognizing languages in $\mathrm{L}(\mathbf{DO} \cap \overline{\mathbf{Ab}})$ it is convenient [TT05b] to make use of their logical characterization given by Corollary 5.16: any such language is definable by an $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence in which no modular quantifier appears in the scope of another quantifier. We illustrate the upper bound on an example.

Example 6.5.
Consider the language $L = \{b,c\}^*a(\{a,b\}^*c\{a,b\}^*c\{a,b\}^*)^*$ which we already studied in Examples 2.5, 5.1 and 5.18. We saw that L can be defined by the $\mathbf{FO}^+\mathbf{MOD}_2[<]$ sentence

$$\phi : \exists^{0 \bmod 2} x \ [Q_c x \wedge (\exists y \ (y < x) \wedge (Q_a y \wedge \forall x \ [(y < x) \Rightarrow \neg Q_a x]))]$$

We want to build a circuit C with $O(n)$ wires verifying $w \models \phi$. As a first step, we build a subcircuit C_ψ with $O(n)$ outputs which simultaneously computes for all $1 \leq y \leq n$ the boolean value of the subformula

$$\psi(y) : Q_a y \wedge \forall x \ [(y < x) \Rightarrow \neg Q_a x].$$

This subformula is true at y if and only if y contains the first occurrence of a in w. Using Bilardi and Preparata's construction we can build a subcircuit with $O(n)$ wires and n outputs which simultaneously tells us for each y if the suffix following y contains an a and this allows the construction of C_ψ.

We can now use the same idea to build a subcircuit C_η with $O(n)$ wires and n output gates which uses the outputs of C_ψ as inputs in order to compute simultaneously for all x the value of

$$\eta(x) : Q_c x \wedge (\exists y \ [(y < x) \wedge \psi(y)]) .$$

Finally, we complete the construction of our circuit C by feeding the n outputs of C_η into a MOD_2 output gate for C.

This example has a straightforward generalization providing the upper bound for all regular languages in $\mathrm{L}(\mathbf{DO} \cap \overline{\mathbf{Ab}})$. We know from Theorem 6.3 that a language K is computable by a family of ACC^0 circuits with $O(n)$ gates if and only if it K is $\mathbf{FO}^+\mathbf{MOD}_2$ definable given arbitrary numerical predicates but there is no similar logical characterization for the class of ACC^0 circuits with $O(n)$ wires. Theorem 6.4 indicates that the fine line separating $O(n)$ gates and $O(n)$ wires may be related to the ability or incapacity of pulling out modular quantifiers in $\mathbf{FO}^+\mathbf{MOD}_2$ sentences.

7. Conclusion

We believe that the block-product/substitution principle largely explains the success of semigroup theory in the analysis of the expressive power of fragments of $\mathbf{FO}^+\mathbf{MOD}[<]$ and **LTL**. In particular, we have tried to show that it underlies some of the most important results about the expressivity of fragments of $\mathbf{FO}^+\mathbf{MOD}[<]$ because it translates these logical questions into algebraic questions about decomposition of pseudovarieties through iterated block-products.

Considerable efforts have been invested in the development of an analogous algebraic approach to regular tree-languages. There currently exists no known algorithm for deciding whether a tree language is definable in $\mathbf{FO}[<]$ where $<$ is the descendant relation in trees. While most agree that this question will inevitably be solved using *some* algebraic framework, it is rather unclear what the correct framework is. For instance, one

can define the syntactic monoid of a regular-tree language L as the transition monoid of the minimal tree-automaton for L. It is known that if a regular tree language is $\mathbf{FO}[<]$-definable then its syntactic monoid is aperiodic but that condition is known to be insufficient [Heu91, PT93]. This strongly suggests that the combinatorial properties of regular tree-languages are not properly reflected in the algebraic properties of its syntactic monoid. Ésik and Weil proposed to consider instead syntactic *pre-clones*. They obtain an analog of the block-product/substitution principle and show that a tree language is $\mathbf{FO}[<]$-definable iff its syntactic preclone belongs to the smallest pseudovariety of pre-clones containing a very simple pseudovariety of preclones and closed under block product [ÉW05]. Unfortunately, too little is known about this pseudovariety to make this characterization effective. That the block-product/substitution principle generalizes to more complex settings is not much of a surprise since it simply provides a scheme to reformulate a logical question into algebraic terms but there are no known preclone analogs of the block-product decomposition results that exist for monoids and this impedes progress.

There are decidability results for subclasses of $\mathbf{FO}[<]$ definable tree languages (e.g. [BW04]), some of which rely on the study of *tree algebras* proposed by Wilke [Wil96]. This first led to an effective algebraic characterization of frontier testable tree languages [Wil96] and, more recently, Benedikt and Segoufin used a similar framework to provide an effective algebraic characterization[10] of tree languages definable in $\mathbf{FO}[S]$ (where S is the child relation) [BS05]. The recent results of Bojańczyk et al. on pebble automata [BSSS06] also seem to be tightly connected to some variant of block-products although the authors do not explicitly give an algebraic interpretation of their work.

We focused in this survey on the case where logical sentences are interpreted over finite words. However, Büchi's Theorem also holds for infinite words: an ω-language is ω-regular if and only if it can be defined by an $\mathbf{MSO}[<]$-sentence. The algebraic theory of ω-regular languages is well-developed although not as robust as the one presented here for the case of finite words [PP04]. Still, the class of ω-languages definable in $\mathbf{FO}[<]$ and $\mathbf{LTL}$ are exactly the starfree ω-languages [Tho79, SPW91, Coh91] which, in turn, are exactly those recognizable by aperiodic ω-semigroups [Per83]. Because the results for finite words often extend to the infinite case [Lib04, PP04, Pin96, Pin01, Tho97], it is tempting to overlook the related caveats. It would be interesting to specifically consider how the block-product/substitution principle extends to the case of infinite words to unify the existing results. The work of Carton [Car00] probably provides all the necessary tools for this investigation.

More generally, as Weil clearly demonstrates in [Wei04], there are numerous extensions of the algebraic point of view on finite automata and regular languages which have proved to be successful in the analysis of more sophisticated machines and more sophisticated logical formalisms. These include regular sets of traces [DR95], series-parallel pomsets [Kus03, LW00] and graphs, as well as timed automata [BDM$^+$06, FK03, MP04, BPT03].

Acknowledgements: We want to thank the anonymous referees for their suggestions to improve the readability of the paper. We also want to thank Luc Segoufin and Jean-Éric Pin for useful discussions.

[10] Segoufin has recently acknowledged that the characterization given in the conference paper is incorrect, but the decidability result still stands [Seg] and a corrected manuscript is available from Segoufin's home page.

References

[Ajt83] M. Ajtai. Σ^1_1-formulae on finite structures. *Annals of Pure and Applied Logic*, **24**: 1–48, 1983.

[All97] E. Allender. Circuit complexity before the dawn of the new millennium. Tech. Rep. 97-49, DIMACS, 1997.

[Alm94] J. Almeida. *Finite Semigroups and Universal Algebra*. Series in Algebra, Vol 3. World Scientific, 1994.

[Bar89] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, **38**(1): 150–164, 1989.

[BCST92] D. A. M. Barrington, K. J. Compton, H. Straubing and D. Thérien. Regular languages in NC^1. *J. Comput. Syst. Sci.*, **44**(3): 478–499, 1992.

[BDM+06] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick and L. Segoufin. Two-variable logic on words with data. In *Proc. 21st Symp. on Logic in Comp. Sci. (LICS-06)*, pp. 7–16. 2006.

[BIL+01] D. A. M. Barrington, N. Immerman, C. Lautemann, N. Schweikardt and D. Thérien. The Crane Beach conjecture. In *Proc. 16th Symp. on Logic in Comp. Sci. (LICS-01)*, pp. 187–196. 2001.

[BIS90] D. A. M. Barrington, N. Immerman and H. Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci.*, **41**(3): 274–306, 1990.

[BL06] C. Behle and K.-J. Lange. **FO**-uniformity. In *Proc. 21st Conf. on Computational Complexity (CCC'06)*. 2006.

[BMT99] A. Baziramwabo, P. McKenzie and D. Thérien. Modular temporal logic. In *Proc. 15th Conf. on Logic in Comp. Sci. (LICS'99)*. 1999.

[Bou05] J. Bourgain. Estimations on certain exponential sums arising complexity theory. *C. R. Acadmie des Sciences Paris I*, **340**: 627–631, 2005.

[BP90] G. Bilardi and F. P. Preparata. Characterization of associative operations with prefix circuits of constant depth and linear size. *SIAM J. Comput.*, **19**(2): 246–255, 1990.

[BPT03] P. Bouyer, A. Petit and D. Thérien. An algebraic characterization of data and timed languages. *Information and Computation*, **182**: 137–162, 2003.

[BS94] D. A. M. Barrington and H. Straubing. Complex polynomials and circuit lower bounds for modular counting. *Computational Complexity*, **4**(4): 325–338, 1994.

[BS95] D. A. M. Barrington and H. Straubing. Superlinear lower bounds for bounded-width branching programs. *J. Comput. Syst. Sci.*, **50**(3): 374–381, 1995.

[BS99] D. A. M. Barrington and H. Straubing. Lower bounds for modular counting by circuits with modular gates. *Computational Complexity*, **8**(3): 258–272, 1999.

[BS05] M. Benedikt and L. Segoufin. Regular tree languages definable in FO. In *Proc. 22nd Symp. on Theoretical Aspects of Comp. Sci. (STACS'05)*, pp. 327–339. 2005.

[BSSS06] M. Bojańczyk, M. Samuelides, T. Schwentick and L. Segoufin. On the expressive power of pebble automata. In *Proc. Int. Coll. on Automata Languages and Programming (ICALP'06)*. 2006.

[BST90] D. A. M. Barrington, H. Straubing and D. Thérien. Non-uniform automata over groups. *Information and Computation*, **89**(2): 109–132, 1990.

[BT88] D. A. M. Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, **35**(4): 941–952, 1988.

[Büc60] J. Büchi. Weak second order arithmetic and finite automata. *Z. Math. Logik und Grundl. Math.*, **6**: 66–92, 1960.

[BW04] M. Bojańczyk and I. Walukiewicz. Characterizing ef and ex tree logics. In *Proc. 15th Concurrency Theory (CONCUR'04)*, pp. 131–145. 2004.

[Car00] O. Carton. Wreath product and infinite words. *J. Pure and Applied Algebra*, **153**: 129–150, 2000.

[CB71] R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *J. Comput. Syst. Sci.*, **5**(1): 1–16, 1971.

[CFL85] A. K. Chandra, S. Fortune and R. J. Lipton. Unbounded fan-in circuits and associative functions. *J. Comput. Syst. Sci.*, **30**(2): 222–234, 1985.

[CH91] S. Cho and D. T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theor. Comput. Sci.*, **88**(1): 99–116, 1991.

[CKK+07] A. Chattopadhyay, A. Krebs, M. Koucký, M. Szegedy, P. Tesson and D. Thérien. Functions with bounded multiparty communication complexity, 2006. Submitted.

[Coh91] J. Cohen. On the expressive power of temporal logic for infinite words. *Theoretical Computer Science*, **83**: 301–312, 1991.

[CPP93] J. Cohen, D. Perrin and J.-E. Pin. On the expressive power of temporal logic. *Journal of Computer and System Sciences*, **46**(3): 271–294, 1993.

[CPS06] L. Chaubard, J.-É.. Pin and H. Straubing. First order formulas with modular predicates. In *Proc. 21st Symp. on Logic in Comp. Sci. (LICS'06)*. 2006.

[CS01] K. J. Compton and H. Straubing. Characterizations of regular languages in low level complexity classes. In *Current Trends in Theoretical Computer Science*, pp. 235–246. 2001.

[CSV84] A. K. Chandra, L. J. Stockmeyer and U. Vishkin. Constant depth reducibility. *SIAM J. Comput.*, **13**(2): 423–439, 1984.

[DR95] V. Diekert and G. Rozenberg, eds. *The book of Traces*. World Scientific, 1995.

[Eil76] S. Eilenberg. *Automata, Languages and Machines*, vol. B. Academic Press, 1976.

[EVW97] K. Etessami, M. Vardi and T. Wilke. First-order logic with two variables and unary temporal logic. In *Proc. 12th IEEE Symposium on Logic in Computer Science*, pp. 228–235. 1997.

[EVW02] K. Etessami, M. Y. Vardi and T. Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, **179**(2): 279–295, 2002.

[EW00] K. Etessami and T. Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Inf. Comput.*, **160**(1-2): 88–108, 2000.

[ÉW05] Z. Ésik and P. Weil. Algebraic recognizability of regular tree languages. *Theor. Comput. Sci.*, **340**(1): 291–321, 2005.

[FK03] N. Francez and M. Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theor. Comput. Sci.*, **306**(1-3): 155–175, 2003.

[FSS84] M. Furst, J. B. Saxe and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, **17**(1): 13–27, 1984.

[GL84] Y. Gurevich and H. Lewis. A logic for constant-depth circuits. *Information and Control*, **61**(1): 65–74, 1984.

[GRS05] F. Green, A. Roy and H. Straubing. Bounds on an exponential sum arising in boolean circuit complexity. *C. R. Acadmie des Sciences Paris I*, **341**: 279–282, 2005.

[GS00] C. Glaßer and H. Schmitz. Languages of dot-depth 3/2. In *Proc. 17th Symp. on Theoretical Aspects of CS (STACS'00)*, pp. 555–566. 2000.

[GS01] C. Glaßer and H. Schmitz. Level 5/2 of the straubing-thérien hierarchy for two-letter alphabets. In *Developments in Language Theory (DLT'01)*, pp. 251–261. 2001.

[Heu91] U. Heuter. First-order properties of trees, star-free expressions and aperiodicity. *Informatique Théorique et Applications*, **25**: 125–146, 1991.

[IK89] N. Immerman and D. Kozen. Definability with a bounded number of bound variables. *Information and Computation*, **83**: 121–13, 1989.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, **16**(4): 760–778, 1987.

[Kam68] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. Ph.D. thesis, University of California, Berkeley, 1968.

[KLPT06] M. Koucký, C. Lautemann, S. Poloczek and D. Thérien. Circuit lower bounds via Ehrenfeucht-Fraïssé games. In *Proc. 21st Conf. on Computational Complexity (CCC'06)*. 2006.

[KPT05] M. Koucký, P. Pudlák and D. Thérien. Bounded-depth circuits: separating wires from gates. In *Proc. 37th ACM Symp. on Theory of Computing (STOC'05)*, pp. 257–265. 2005.

[KR65] K. Krohn and J. Rhodes. The algebraic theory of machines I. *Trans. Amer. Math. Soc.*, **116**: 450–464, 1965.

[Kus03] D. Kuske. Towards a language theory for infinite n-free pomsets. *Theor. Comput. Sci.*, **1-3**(299): 347–386, 2003.

[Lan04] K.-J. Lange. Some results on majority quantifiers over words. In *Proc. 19th Conf. on Computational Complexity (CCC'04)*, pp. 123–129. 2004.

[Lib04] L. Libkin. *Elements of Finite Model Theory*. Springer Verlag, 2004.

[LMSV01] C. Lautemann, P. McKenzie, T. Schwentick and H. Vollmer. The descriptive complexity approach to LOGCFL. *J. Comput. Syst. Sci.*, **62**(4): 629–652, 2001.

[LTT06] C. Lautemann, P. Tesson and D. Thérien. An algebraic point of view on the crane beach property. In *Proc. Comp. Sci. Logic (CSL'06)*, pp. 426–440. 2006.

[LW00] K. Lodaya and P. Weil. Series-parallel languages and the bounded-width property. *Theor. Comput. Sci.*, **237**(1-2): 347–380, 2000.

[Lyn82a] J. F. Lynch. Complexity classes and theories of finite models. *Mathematical Systems Theory*, **15**(2): 127–144, 1982.

[Lyn82b] J. F. Lynch. On sets of relations definable by addition. *J. Symb. Log.*, **47**(3): 659–668, 1982.

[MP71] R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, Cambridge, Mass., 1971.

[MP04] O. Maler and A. Pnueli. On recognizable timed languages. In *Proc. Found. of Software Science and Computation Structures (FoSSaCS'04)*, pp. 348–362. 2004.

[MPT91] P. McKenzie, P. Péladeau and D. Thérien. NC^1: The automata theoretic viewpoint. *Computational Complexity*, **1**: 330–359, 1991.

[MSW01] S. Margolis, M. Sapir and P. Weil. Closed subgroups in the pro-**V** topologies, and the extension problem for inverse automata. *Intern. J. Algebra and Computation*, **11**: 405–445, 2001.

[Nur00] J. Nurmonen. Counting modulo quantifiers on finite structures. *Inf. Comput.*, **160**(1-2): 62–87, 2000.

[Pél92] P. Péladeau. Formulas, regular languages and boolean circuits. *Theor. Comput. Sci.*, **101**(1): 133–141, 1992.

[Per83] D. Perrin. Variétés de semigroupes et mots infinis. In *Proc. 10th Int. Conf. on Automata, Languages and Programming (ICALP'83)*, pp. 610–616. 1983.

[Pin86] J.-E. Pin. *Varieties of formal languages*. North Oxford Academic Publishers Ltd, London, 1986.

[Pin95] J.-É. Pin. $PG = BG$, a success story. In *NATO Advanced Study Institute Semigroups, Formal Languages and Groups*, pp. 33–47. Kluwer academic publishers, 1995.

[Pin96] J.-É. Pin. Logic, semigroups and automata on words. *A. of Math. and Art. Int.*, **16**: 343–384, 1996.

[Pin97] J.-E. Pin. Syntactic semigroups. In *Handbook of language theory*, vol. 1, chap. 10, pp. 679–746. Springer Verlag, 1997.

[Pin01] J.-E. Pin. Logic on words. In *Current Trends in Theoretical Computer Science*, pp. 254–273. 2001.

[PP86] D. Perrin and J.-E. Pin. First-order logic and star-free sets. *J. Comput. Syst. Sci.*, **32**(3): 393–406, 1986.

[PP04] D. Perrin and J. Pin. *Infinite Words*, vol. 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

[PS81] J.-É. Pin and H. Straubing. Monoids of upper triangular matrices. *Colloquia Mathematica Societatis Janos Bolyai, Semigroups, Szeged*, **39**: 259–272, 1981.

[PST97] P. Péladeau, H. Straubing and D. Thérien. Finite semigroup varieties defined by programs. *Theor. Comput. Sci.*, **180**(1-2): 325–339, 1997.

[PT93] A. Potthoff and W. Thomas. Regular tree-languages without unary symbols are star-free. In *Proc. Fund. of Comp. Theory (FCT'93)*, pp. 396–405. 1993.

[Pud94] P. Pudlák. Communication in bounded depth circuits. *Combinatorica*, **14**(2): 203–216, 1994.

[PW97] J. E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory Comput. Systems*, **30**: 383–422, 1997.

[PW01] J.-E. Pin and P. Weil. A conjecture on the concatenation product. *ITA*, **35**(6): 597–618, 2001.

[RS06] A. Roy and H. Straubing. Definability of languages by generalized first-order formulas over (**N**, +). In *23rd Symp. on Theoretical Aspects of Comp. Sci. (STACS'06)*, pp. 489–499. 2006.

[RT89] J. Rhodes and B. Tilson. The kernel of monoid morphisms. *J. Pure and Applied Algebra*, **62**: 227–268, 1989.

[RTT98] J.-F. Raymond, P. Tesson and D. Thérien. An algebraic approach to communication complexity. In *Proc. 25th Int. Coll. on Automata Languages and Programming (ICALP'98)*, pp. 29–40. 1998.

[Sch65] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Computation*, **8**(2): 190–194, 1965.

[Sch76] M. P. Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, **13**: 47–75, 1976.

[Sch05] N. Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Trans. Comput. Log.*, **6**(3): 634–671, 2005.

[Seg] L. Segoufin. Personal communication.

[Sim75] I. Simon. Piecewise testable events. In *Proc. 2nd GI Conf.*, pp. 214–222. 1975.

[Smo86] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proc. 19^{th} ACM STOC*, pp. 77–82. 1986.

[SN97] A. P. Stolboushkin and D. Niwinski. y = 2x vs. y = 3x. *J. Symb. Log.*, **62**(2): 661–672, 1997.

[SPW91] B. L. Saëc, J.-E. Pin and P. Weil. A purely algebraic proof of McNaughton's theorem on infinite words. In *11th Conf. Found. of Software Tech. and Theor. Comp. Sci. (FSTTCS'91)*, pp. 141–151. 1991.

[ST02] 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.

[ST03] 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.

[ST06] H. Straubing and D. Thérien. A note on MOD_p-MOD_m-circuits. *Theory of Computing Systems*, **39**(5): 699–706, 2006.

[Str85] H. Straubing. Finite semigroup varieties of the form $\mathbf{V} * \mathbf{D}$. *J. Pure and Applied Algebra*, **36**: 53–94, 1985.

[Str88] H. Straubing. Semigroups and languages of dot-depth two. *Theor. Comput. Sci.*, **58**: 361–378, 1988.

[Str92] H. Straubing. Circuit complexity and the expressive power of generalized first-order formulas. In *Proc. 19th Int. Conf. on Automata, Languages and Programming (ICALP'92)*, pp. 16–27. 1992.

[Str94] H. Straubing. *Finite Automata, Formal Logic and Circuit Complexity*. Boston: Birkhauser, 1994.

[Str00] H. Straubing. When can one monoid simulate another? In *Algorithmic Problems in Groups and Semigroups*, pp. 267–288. Birkhäuser, 2000.

[Str02] H. Straubing. On the logical description of regular languages. In *Proc. of the 5th Latin American Theoretical Informatics Conference (LATIN '02)*. 2002.

[STT93] H. Straubing, D. Thérien and W. Thomas. Logics for regular languages, finite monoids and circuit complexity. In *NATO Advanced Study Institute Semigroups, Formal Languages and Groups*. Kluwer academic publishers, 1993.

[STT95] H. Straubing, D. Thérien and W. Thomas. Regular languages defined by generalized quantifiers. *Information and Computation*, **118**: 289–301, 1995.

[SW92] H. Straubing and P. Weil. On a conjecture concerning dot-depth two languages. *Theor. Comput. Sci.*, **104**(2): 161–183, 1992.

[Thé94] D. Thérien. Circuits constructed with MOD_q gates cannot compute AND in sublinear size. *Computational Complexity*, **4**: 383–388, 1994.

[Tho79] W. Thomas. Star-free regular sets of ω-sequences. *Information and Control*, **42**(2): 148–156, 1979.

[Tho82] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, **25**(3): 360–376, 1982.

[Tho97] W. Thomas. *Languages, Automata and Logic*, vol. III, chap. 7, pp. 389–455. Springer, 1997.

[Til87] B. Tilson. Categories as algebras. *J. Pure and Applied Algebra*, **48**: 83–198, 1987.

[TT02] P. Tesson and D. Thérien. Diamonds are forever: the variety **DA**. In *Semigroups, Algorithms, Automata and Languages*. WSP, 2002.

[TT04] P. Tesson and D. Thérien. Monoids and computation. *International Journal on Algebra and Computation*, **14**(5–6): 801–816, 2004.

[TT05a] P. Tesson and D. Thérien. Complete classifications for the communication complexity of regular languages. *Theory of Computing Systems*, **38**(2): 135–159, 2005.

[TT05b] P. Tesson and D. Thérien. Restricted two-variable sentences, circuits and communication complexity. In *Proc. 32nd Int. Conf. on Automata, Languages and Programming (ICALP'05)*, pp. 526–538. 2005.

[TT06] P. Tesson and D. Thérien. Bridges between algebraic automata theory and complexity theory. *The Complexity Column, Bull. EATCS*, **88**: 37–64, 2006.

[TW85] D. Thérien and A. Weiss. Graph congruences and wreath products. *J. Pure and Applied Algebra*, **36**: 205–215, 1985.

[TW98] 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*, pp. 256–263. 1998.

[TW02] D. Thérien and T. Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. *SIAM Journal on Computing*, **31**, 2002.

[TW04] D. Thérien and T. Wilke. Nesting until and since in linear temporal logic. *Theory Comput. Syst.*, **37**(1): 111–131, 2004.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Texts in theoretical computer science. Springer, 1999.

[Wei89] P. Weil. Inverse monoids of dot-depth two. *Theor. Comput. Sci.*, **66**(3): 233–245, 1989.

[Wei92] P. Weil. Closure of varieties of languages under products with counter. *J. Comput. Syst. Sci.*, **45**: 316–339, 1992.

[Wei04] P. Weil. Algebraic recognizability of languages. In *Proc. 29th Int. Symp. Math. Found. of Comp. Sci. (MFCS'04)*, pp. 149–175. 2004.

[Wil96] T. Wilke. An algebraic characterization of frontier testable tree languages. *Theor. Comput. Sci.*, **154**(1): 85–106, 1996.

[Wil01] T. Wilke. Linear temporal logic and finite semigroups. In *26th Int. Symp. on Math. Found. of Comp. Sci. (MFCS'01)*, pp. 96–110. 2001.