On the Complexity of Free Monoid Morphisms

Klaus-Jörn Lange and Pierre McKenzie*

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
{lange,mckenzie}@informatik.uni-tuebingen.de

Abstract. We locate the complexities of evaluating, of inverting, and of testing membership in the image of, morphisms $h : \Sigma^* \to \Delta^*$. By and large, we show these problems complete for classes within NL. Then we develop new properties of finite codes and of finite sets of words, which yield image membership subproblems that are closely tied to the unambiguous space classes found between L and NL.

1 Introduction

Free monoid morphisms $h : \Sigma^* \to \Delta^*$, for finite alphabets Σ and Δ, are an important concept in the theory of formal languages (e.g. [6, 11]), and they are relevant to complexity theory. Indeed, it is well known (e.g. [7]) that $\mathrm{NP} = \mathit{Closure}(\leq_m^{\mathrm{AC}^0}, HOM_{n.e.}) \subset \mathit{Closure}(\leq_m^{\mathrm{AC}^0}, HOM) = R.E.$, where $\leq_m^{\mathrm{AC}^0}$ denotes many-one AC^0-reducibility, HOM (resp. $HOM_{n.e.}$) is the set of morphisms (resp. nonerasing morphisms), and *Closure* denotes the smallest class of languages containing a finite nontrivial language and closed under the relations specified. Morphisms and their inverses also play a role in studying regular languages and "small" complexity classes: regular language varieties are closed under inverse morphisms [6], and the replacement of morphisms by "polynomial length M-programs", in the definition of recognition by a finite monoid M, allows automata to capture many subclasses of NC^1 [3, 5, 13].

Here we consider the complexity of evaluating inverse morphisms and morphisms $h : \Sigma^* \to \Delta^*$. Specifically, we consider the simple problem **eval** of computing the image of a word v under h, the problem **range** of determining whether a word $w \in h(\Sigma^*)$, and the problem **inv** of computing an element of $h^{-1}(w)$ given $w \in h(\Sigma^*)$. We examine the *fixed* setting, in which the morphism is input-independent, and the *variable* setting, in which the morphism is defined as part of the input.

The general framework of our results is summarized in the following figure. In the fixed case, the **eval** problem characterizes the relation between the classes NC^0, AC^0 and TC^0, and the problems **range** and **inv** are

* On sabbatical leave from the Université de Montréal until August 1998.

closely related to the class NC^1. Membership of **inv** in NC^1 is then to be contrasted with Håstad's result that there exists a fixed NC^0-computable function whose associated inversion problem is P-complete [10]. We also observe that $\mathit{Closure}(\leq_T^{\mathrm{AC}^0}, HOM^{-1}) = \mathrm{TC}^0$, where HOM^{-1} denotes inverse morphisms, yielding yet another characterization of this important subclass of NC^1. In the variable case, the problem **eval** remains in TC^0 while the **range** and **inv** problems capture complexity classes between L and NL.

Problem	*Fixed* setting	*Variable* setting
evaluation	isometric: NC^0 nonisom.: TC^0-complete	isometric: AC^0 nonisom.: TC^0-complete
range	any h: NC^1 chosen h: NC^1-complete [1]	$h(\Sigma)$ prefix code: L-complete unrestricted h: NL-complete
inversion	NC^1	$(\text{Functional-L})^{NL}$

Here we do not distinguish between a circuit-based language class and its functional counterpart. A morphism $h : \Sigma^* \to \Delta^*$ is isometric iff h applied to each $a \in \Sigma$ yields a word of the same length.

An important part of our results, motivated by recent interest in classes intermediate between L and NL [2, 12, 14], is the investigation of the problem **range** in the variable case. Restricting the underlying morphism affects the complexity of the **range** problem, e.g. the **range** problem for prefix codes is in L and is complete for this class. Now, one might expect that imposing the code property on $h(\Sigma)$ should render the **range** problem complete for an unambiguous logspace class. However the resulting problem remains NL-complete. We therefore develop properties of codes and of sets of words, in particular the *stratification* property (see Section 4.2), which yield **range** subproblems of complexity identical to the complexity of the graph accessibility problems introduced to study unambiguous logspace classes (see Section 2). In particular, we show that the problem **range** in which $h(\Sigma)$ is a *stratified code* is many-one equivalent to the GAP problem L_{stu} capturing $\mathit{StUSPACE}(\log n)$, and that a variant of **range** in which $h(\Sigma)$ is a *stratified left partial* code is $\mathit{RUSPACE}(\log n)$-complete. This is particularly interesting since $\mathit{RUSPACE}(\log n)$ was only recently found to have a complete problem [12].

Due to space restrictions many of our constructions cannot be given in detail. In particular, the proofs of corollary 3 and theorems 4, 7, 8, 9, and 10 have to be postponed to a full version of this paper.

2 Preliminaries

2.1 Complexity theory

We assume familiarity with basic complexity theory. In particular, recall $\mathrm{NC}^0 \subset \mathrm{AC}^0 \subset \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathit{StUSPACE}(\log n) \subseteq \mathit{RUSPACE}(\log n) \subseteq \mathit{UL} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP}$, where UL is the set of languages accepted by logspace Turing machines which are nondeterministic with at most one accepting computation, $\mathit{RUSPACE}(\log n)$ is defined like UL with the stronger condition that, on any input, at most one path should exist from the initial configuration to *any* accessible configuration y, and $\mathit{StUSPACE}(\log n)$ is defined like UL with yet the stronger condition that, between any pair of configurations (x, y), at most one path should exist from x to y [12]. Furthermore, Functional-L, also denoted FL, is the functional counterpart of L, i.e. the set of functions computable by deterministic Turing machines and FL^{NL} is the set of functions computable by a deterministic Turing machine M having access to an NL-oracle.

In the usual way, DLOGTIME uniformity of a circuit family refers to the ability for a Turing machine equipped with an index tape allowing direct access to its input to recognize in time $O(\log n)$ the extended connectivity language of a circuit. For precise details on circuit descriptions, see [4].

Just as GAP is NL-complete and outdegree-one GAP is L-complete [9], the obvious GAP problems L_{stu}, L_{ru}, and L_u, which are respectively $\mathit{StUSPACE}(\log n)$-hard, $\mathit{RUSPACE}(\log n)$-hard, and UL-hard, were introduced for topologically sorted graphs in [2, 12]. L_{ru} is $\mathit{RUSPACE}(\log n)$-complete [12] while L_{stu} is not known to belong to $\mathit{StUSPACE}(\log n)$. Using the closure under complement of NL L_u is already NL-complete.

We will make use of the reducibilities $\leq_m^{\mathrm{AC}^0}$ and $\leq_T^{\mathrm{AC}^0}$, which refer to many-one and Turing AC^0 reducibilities respectively.

2.2 Problem definitions

Fix a morphism $h : \Sigma^* \to \Delta^*$ for finite alphabets Σ and Δ, with Σ assumed ordered. In the fixed setting, the three problems of interest in this note are:

eval(h) Given $v \in \Sigma^*$, compute $h(v)$. The *decision problem* has $b \in \Sigma$ and $j \in N$ as further inputs, and asks whether b is the jth symbol in $h(v)$.

range(h) Given $w \in \Delta^*$, determine whether $w \in h(\Sigma^*)$.

inv(h) Given $w \in \Delta^*$, express w as $h(a_{i_1})h(a_{i_2})\cdots h(a_{i_k})w'$ such that, first, $|w'|$ is minimal, and second, $v := a_{i_1}a_{i_2}\cdots a_{i_k}$ is lexicographically minimal with respect to the ordering of Σ. The *decision problem* is obtained by adding $j \in N$ as an input parameter, and asking for the jth bit in the representation of w.

In the variable setting, the three problems of interest are **eval**, **range**, and **inv**, defined as above, except that the alphabets Σ and Δ, and the morphism h, now form part of the input.

Fix a finite alphabet Γ. Let $v = a_1a_2\cdots a_n, a_i \in \Gamma, n \geq 0$. The length of v is written $|v|$, and $\#_a(v)$ represents the number of occurrences of $a \in \Gamma$ in v. For $0 \leq i \leq j \leq n$, we define ${}_iv_j$ as $a_{i+1}\cdots a_j$. In particular, ${}_{i-1}v_i$ is a_i if $i \geq 1$.

We encode our problems, in both the fixed and the variable settings, as binary strings. In the fixed setting, a word v over an alphabet Γ is encoded as a sequence of equal length "bytes", each representing a letter in Γ. By the predicate $Q_a(v,i)$, $a \in \Gamma$, $v \in \Gamma^*$, we mean that ${}_{i-1}v_i$ is a (with $Q_a(v,i)$ false when $i > |v|$). In the variable setting, we write $Q_a(v,i)$ as $Q(a,v,i)$ in view of the input-dependent alphabet Σ. Any encoding which allows computing $Q(a,v,i)$ in AC^0 suffices.

To encode a morphism $h : \Sigma^* \to \Delta^*$ (only required in the variable setting), it suffices that the predicate associated with h, denoted $H(b,a,j)$, where $b \in \Delta$, $a \in \Sigma$, and $j \geq 0$, meaning that ${}_{j-1}h(a)_j$ is b, be AC^0-computable. For $h : \Sigma^* \to \Delta^*$, we define $\max(h)$ as $\max\{|h(a)| : a \in \Sigma\}$. The predicate $H(b,a,j)$ extends to $H(b,v,j)$ for $v \in \Sigma^*$ in the obvious way, and computing this extension is the object of the **eval** problem. We define $H(b,v,j)$ to be false if $j > |h(v)|$.

3 Fixed Case

Throughout this section we fix $h : \Sigma^* \to \Delta^*$. We write the associated predicate $H(b,v,j)$, $b \in \Delta$, $v \in \Sigma^*$, $j \in N$ as $H_b(v,j)$, in view of the fixed alphabet Σ.

3.1 Evaluation

The complexity of evaluating h is low, but it depends in an interesting way on whether h is *isometric*, i.e. whether the length of any $h(w)$ only depends on $|w|$.

Proposition 1. **eval**(h) $\in$ *Functional-NC*^{0} *if h is isometric.*

Proof. Let h be isometric and let $c = |h(a)|$ ($= \max(h)$) for any $a \in \Sigma$. Then, for $v \in \Sigma^*$, $b \in \Delta$, and $j \in N$, we have $H_b(v, j) \Leftrightarrow (\exists a \in \Sigma)[Q_a(v, \lceil j/c \rceil) \wedge H_b(a, j \ mod \ c)]$. Since Σ is fixed, the existential quantification over a can be done in NC^0. Hence, for each j, $1 \leq j \leq c \cdot |v|$, there is an NC^0 subcircuit C_j computing the jth symbol in $h(v)$. The circuit for **eval**(h) is a parallel arrangement of these C_j, $1 \leq j \leq c \cdot |v|$. Uniformity will be argued in a full version of this paper.

It follows that, when h is isometric, the decision problem **eval**$(h) \in$ AC^0. Interestingly, the converse also holds: if h is not isometric, then **eval**$(h) \notin$ Functional-AC^0 and the decision problem **eval**$(h) \notin$ AC^0, as follows from:

Theorem 1. *If h is non-isometric, then the decision problem* **eval***(h) is TC^0-hard under $\leq_T^{AC^0}$.*

Proof. Since h is not isometric, there exist $a, b \in \Sigma$, $s, t \in N$, such that $|h(a)| = s$ and $|h(b)| = s + t$ with $t > 0$. We claim that MAJORITY, i.e. the language of all words v over the alphabet $\{a, b\}$ having $|v|/2 \leq \#_b(v)$, $\leq_T^{\mathrm{AC}^0}$-reduces to computing the length of $h(v)$. This implies that the extended predicate $H_b(v, j)$, $v \in \Sigma^*$, is TC^0-hard, because $|h(v)|$ is trivially expressed as the largest j, $1 \leq j \leq |v| \cdot \max(h)$, such that $\bigvee_{b \in \Sigma} H_b(v, j)$. And computing the extended predicate $H_b(v, j)$ is precisely the decision problem **eval**(h).

To prove the claim that MAJORITY reduces to computing $|h(v)|$, note that $|v|/2 \leq \#_b(v)$ iff $s \cdot |v| + t \cdot |v|/2 \leq s \cdot |v| + t \cdot \#_b(v)$. Now the left hand side of the latter equality can be computed in AC^0 because s and t are constants. As to the right hand side, it is precisely $|h(v)| = s \cdot \#_a(v) + (s + t) \cdot \#_b(v) = s \cdot |v| + t \cdot \#_b(v)$. Hence one can test $|v|/2 \leq \#_b(v)$ in AC^0, once an oracle gate provides the value of $|h(v)|$. Hence MAJORITY $\leq_T^{\mathrm{AC}^0}$-reduces to computing $|h(v)|$.

Corollary 1. *The decision problem* **eval**(h) *is in AC^0 iff h is isometric.*

On the other hand, it can be shown that even a variable morphism can always be evaluated in TC^0, see subsection 4.1. Hence, the reduction from MAJORITY to **eval**(h) exhibited in Theorem 1 can in fact be reversed:

Theorem 2. *For each morphism h, the decision problem* **eval**$(h) \in TC^0$.

Corollary 2. *If h is a non-isometric morphism, then the decision problem* **eval**(h) *is TC^0-complete under $\leq_T^{AC^0}$.*

Corollary 3. $TC^0 = Closure(\leq_T^{AC^0}, HOM^{-1})$.

3.2 Range membership

We now turn to the problem of testing whether a word $w \in \Delta^*$ belongs to $h(\Sigma^*)$. Since Σ^* is regular and the regular languages are closed under morphisms, $h(\Sigma^*)$ is regular. Hence $\mathbf{range}(h) \in \mathrm{NC}^1$. On the other hand, Schützenberger in 1965 constructed the following family of finite biprefix codes over the binary alphabet $\{a, b\}$: $C_n := \{a^n, a^{n-1}ba, a^{n-2}b, ba^{n-1}\} \cup \{a^i ba^{n-i-1} \mid 1 \leq i \leq n-3\}$. Schützenberger [16] proved that, for each n, the symmetric group S_n divides (i.e. is an epimorphic image of a submonoid of) the syntactic monoid of the Kleene closure C_n^* of C_n. Using Theorem IX.1.5 of [17] we get the following result. Again, details are left to to a full version of this paper.

Theorem 3. *For every morphism h,* $\mathbf{range}$*$(h) \in NC^1$. Furthermore, there exists a morphism h such that* $\mathbf{range}$*(h) is NC^1-complete under $\leq_m^{AC^0}$.*

3.3 Inversion

Recall the definition of problem $\mathbf{inv}(h)$. By using $\mathbf{inv}(h)$ to determine whether $|w'| > 0$, the decision problem $\mathbf{range}(h)$ reduces to the decision problem $\mathbf{inv}(h)$, under $\leq_m^{\mathrm{AC}^0}$ if an obvious encoding is chosen. Hence there is an h such that the decision problem $\mathbf{inv}(h)$ is NC^1-hard (by Theorem 3). Our goal in this section is to show that $\mathbf{inv}(h)$ is in Functional-NC^1, which holds, clearly, iff the decision problem $\mathbf{inv}(h)$ is in NC^1.

Suppose that a word $w \in h(\Sigma^*)$. Then if $h(\Sigma)$ were a code, i.e. if it had the property that any word in $h(\Sigma^*)$ is uniquely expressible as a sequence of elements from $h(\Sigma)$, then computing $h^{-1}(w)$ would simply require computing all the positions i, $1 \leq i \leq |w|$, at which w splits into $w = \alpha\beta$ with $\alpha, \beta \in h(\Sigma^*)$. This set of positions would uniquely break w up into words from $h(\Sigma)$. Since these positions can be computed in NC^1 because $h(\Sigma^*)$ is a regular language, this strategy would solve such instances of $\mathbf{inv}(h)$ in NC^1. Interestingly, we can combine this strategy with a greedy approach and solve the general case in which $h(\Sigma)$ is not a code. For lack of space we have omitted the proof of the next theorem.

Theorem 4. *For every morphism h,* $\mathbf{inv}$*(h) is in Functional-NC^1.*

4 Variable Case

4.1 Evaluation

The fixed case construction for $\mathbf{eval}(h)$ carries over to the variable case $\mathbf{eval}$. We must compute the predicate $H(b, v, j)$, i.e. determine whether the jth symbol in $h(v)$ is b. In the following let $\pi(v)$ be the *Parikh* vector of a word v and M_h the growth matrix of the morphism h, i.e. $(M_h)_{ik}$ is the number of b_k symbols in $h(a_i)$. Then, $H(b, v, j)$ holds iff there exists $i \in N$ and $a \in \Sigma$ such that $\pi({}_0v_{i-1})M_h\mathbf{1} < j \leq \pi({}_0v_i)M_h\mathbf{1}$ and $Q(a, v, i)$ and $h(a)_{j-(\pi({}_0v_{i-1})M_h\mathbf{1})} = b$. Now the computation of M_h out of h and the computations of $\pi({}_0v_{i-1})M_h\mathbf{1}$ and $\pi({}_0v_i)M_h\mathbf{1}$ out of v can be done in TC^0. Indeed, the integers involved in these computations are *small*, i.e. their absolute values are polynomial in the input size, and arithmetic with such integers can even be carried out in AC^0 [4, Lemma 10.5]. But the computation of the *Parikh* vectors needs counting and is TC^0-hard. Hence computing $H(b, v, j)$ is TC^0-hard and **eval** $\in$ Functional-TC^0, which we record as:

Theorem 5. *The decision problem* **eval** *is TC^0–complete w.r.t. $\leq_T^{AC^0}$.*

We note that **eval** restricted to isometric morphisms is in AC^0.

4.2 Range membership

Our problem **range** is precisely the *Concatenation Knapsack Problem* considered by Jenner [8], who showed that:

Theorem 6. *(Jenner)* **range** *is NL–complete.*

Recall that a nonempty set $C \subset \Delta^*$ is a *code* if C freely generates C^*, and that C is a *prefix code* if C has the prefix property, i.e. if no word in C has a proper prefix in C. Define **prefixcoderange** to be the **range** problem restricted to input morphisms $h : \Sigma^* \to \Delta^*$ such that $h(\Sigma)$ is a prefix code. The easy proof of the following will be given in a full version of this paper.

Theorem 7. **prefixcoderange** *is L–complete.*

In view of Theorems 6 and 7, it is natural to ask for properties of $h(\Sigma)$ allowing the **range** problem to capture concepts between L and NL like symmetry or unambiguity. For instance, one might reasonably expect *codes* to capture unambiguity, and thus the obvious problem **coderange** to be complete for one of the unambiguous space classes between L and NL. But the mere problem of testing for the code property is NL-hard as shown by Rytter [15]. In the following subsection, we introduce a more elaborate approach which is able to capture unambiguity.

4.2.1 Stratified sets of words and morphisms

Definition 1. **a)** *Let $C \subseteq \Delta^*$ be arbitrary. Define a relation $\rho_C \subseteq \Delta \times \Delta$ as $a \; \rho_C \; b$ iff some word in C contains ab as a subword.*

b) *C is said to be* stratified *if ρ_C forms a unique maximal acyclic chain $a_1 \; \rho_C \; a_2 \; \rho_C \; \cdots \; \rho_C \; a_n$; in that case, the word $\sigma(C) := a_1 a_2 \cdots a_n \in \Delta^*$ is called the* stratification *of C. (Example: The set $\{a, b, ab, bc\} \subset \{a, b, c\}^*$ is stratified, while $\{a, b\}$, $\{ab, ba\}$ and $\{ab, ac\}$ are not.)*

Let $C \subset \Delta^*$ be stratified, where we assume from now on that such a stratified set makes use of all the letters in Δ. Clearly, any word $x \in C$ is a subword of $\sigma(C)$. Then, any word $w \in \Delta^*$ is uniquely expressible as $w = \alpha_1 \alpha_2 \ldots \alpha_k$, where each α_i is a maximal common subword of w and $\sigma(C)$.

Proposition 2. *Let $C \subset \Delta^*$ be a stratified set.*

a) *For any $w \in \Delta^*$, $w \in C^*$ iff its canonical decomposition $w = \alpha_1 \alpha_2 \ldots \alpha_k$ into maximal subwords common with $\sigma(C)$ satisfies $\alpha_i \in C^*$ for each i.*

b) *C is a code iff each subword of its stratification $\sigma(C)$ is expressible in at most one way as a concatenation of words from C.*

Testing whether a finite set $C \subset \Delta^*$ is stratified is L-complete. On the other hand, although stratification may seem like an overwhelming restriction, **stratifiedrange**, i.e. the **range** problem for stratified $h(\Sigma)$, is NL-hard:

Theorem 8. **stratifiedrange** *is NL-complete.*

Proof. To see that **stratifiedrange** is in NL, we first test deterministically in log space whether C is stratified. Then we use the fact that **range** $\in$ NL. The construction used to show NL-hardness is strongly inspired by a proof in [7] but has to be omitted because of lack of space.

Hence, like **coderange**, but for a different reason, **stratifiedrange** is NL-complete and thus does not appear to capture an unambiguous logspace class. It is **stratifiedcoderange**, namely the problem **range** restricted to the case of a stratified code $h(\Sigma)$, which bears a tight relationship to *StUSPACE*$(\log n)$. Again, we have to postpone the proof of the following theorem to a full version of this paper.

Theorem 9. **stratifiedcoderange** *and L_{stu} are many-one logspace equivalent.*

Hence **stratifiedcoderange** is $StUSPACE(\log n)$-hard, and although we are unable to claim a complete problem for $StUSPACE(\log n)$, we have come very close, since even the $StUSPACE(\log n)$-hard language L_{stu}, specifically tailored to capture $StUSPACE(\log n)$, is only known to be in $RUSPACE(\log n)$ [12]. On the other hand, we can claim a complete problem for the unambiguous class $RUSPACE(\log n)$, as follows.

Proposition 2 states that the code property of a stratified set C translates into the unique expressibility of every expressible subword of the stratification of C as an element of C^*. Let us relax this condition and define a stratified set C to be a *left partial code* if no prefix of $\sigma(C)$ can be expressed in two ways as an element of C^*. Furthermore, we let **lpcstratification** be the special case of the **leftpartialcoderange** problem in which we are only asked to determine whether $\sigma(h(\Sigma)) \in h(\Sigma^*)$, i.e. **lpcstratification** is the language of all morphisms $h : \Sigma^* \to \Delta^*$ such that $h(\Sigma)$ is a left partial code and the stratification of $h(\Sigma)$ is in $h(\Sigma^*)$. Then

Theorem 10. **lpcstratification** *is* $RUSPACE(\log n)$*-complete.*

Proof sketch. The construction used in Theorems 8 and 9 in effect proves that **lpcstratification** and L_{ru} are many-one logspace equivalent. Here L_{ru} is the language of graphs having a path from source to target, but required to possess the unique path property only from the source to any accessible node. This is precisely the property which corresponds to the action of canonically expressing the stratification of a left partial code. We then appeal to the main result of [12], namely that L_{ru} is $RUSPACE(\log n)$-complete.

4.3 Inversion

The class (Functional-L)^{NL} can be defined equivalently as the set of functions computed by single valued NL–transducers. We have:

Theorem 11. *Problem* **inv** *is in (Functional-L)*^{NL}.

Proof. To solve **inv**, we use the algorithm and the automaton constructed in Theorem 4. Here, instead of first producing $\widehat{w}$, we start the deterministic logspace simulation of the automaton. Whenever the simulation would require reading a letter from $\widehat{w}$, we appeal to an NL-oracle to test whether the current input position breaks w up into a prefix and a suffix in $h(\Sigma^*)$ (an NL-oracle can test this by Theorem 6). This concludes the proof.

We remark that the deterministic version of **inv**, namely the restriction of **inv** in which h is a prefix code, belongs to Functional-L.

Since computing the **inv** function allows answering the **range** question, we have $\mathrm{NL} \subseteq \mathrm{L}^{\mathbf{inv}}$, and by the previous theorem $\mathrm{NL} = \mathrm{L}^{\mathbf{inv}}$. This implies that $\mathrm{FL}^{\mathrm{NL}} = \mathrm{FL}^{\mathbf{inv}}$, i.e. that the **inv** function is Turing-complete for $\mathrm{FL}^{\mathrm{NL}}$.

References

1. E. Allender, V. Arvind, and M. Mahajan, *Arithmetic Complexity, Kleene Closure, and Formal Power Series,* DIMACS TechRep. 97-61, September 1997.
2. E. Allender and K.-J. Lange, *StUSPACE*$(\log n) \subseteq$ *DSPACE*$(\log^2 n/\log\log n)$, *Proc. of the 7th ISAAC*, Springer LNCS vol. 1178, pp. 193–202, 1996.
3. D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. System Sci.*, 38:150–164, 1987.
4. D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within $\mathcal{NC}^1$. *Journal of Computer and System Sciences*, 41:274–306, 1990.
5. D. Barrington and D. Thérien, Finite Monoids and the Fine Structure of NC^1, *J. of the ACM*, 35(4):941-952, 1988.
6. S. Eilenberg, *Automata, Languages and Machines,* Academic Press, Vol. B, (1976).
7. P. Flajolet and J. Steyaert, *Complexity of classes of languages and operators*, Rap. de Recherche 92 de l'IRIA Laboria, novembre 1974.
8. B. Jenner, Knapsack problems for NL, *Information Processing Letters* 54 (1995), pp. 169-174.
9. N. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975.
10. J. Håstad, *Computational Limitations for Small-Depth Circuits,* PhD Thesis, M.I.T., ACM Doctoral Dissertation Awards, MIT Press (1987).
11. T. Harju and J. Karhumäki, Morphisms, in *Handbook of Formal Languages*, ed. G. Rozenberg and A. Salomaa, Springer, 1997.
12. K.-J. Lange, An unambiguous class possessing a complete set, *Proc. 14th Annual Symp. on Theoret. Aspects of Computer Science,* Springer LNCS vol. 1200, pp. 339–350, 1997.
13. P. McKenzie, P. Péladeau and D. Thérien, NC^1: The Automata-Theoretic Viewpoint, *Computational Complexity* 1:330-359, 1991.
14. K. Reinhardt and E. Allender, Making nondeterminism unambiguous, *Proc. of the 38th IEEE FOCS* , pp. 244–253, 1997.
15. W. Rytter, The space complexity of the unique decipherability problem, *Information Processing Letters* 23 (1986), pp. 1–3.
16. M. Schützenberger, On finite monoids having only trivial subgroups, *Information and Control* 8, 190–194, 1965.
17. H. Straubing, *Finite automata, formal logic, and circuit complexity*, Birkhäuser, Boston, 1994.