Separating the Classes of Recursively Enumerable Languages Based on Machine Size

Jan van Leeuwen

Jiří Wiedermann

Technical Report UU-CS-2014-014
March 2014

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
Princetonplein 5
3584 CC Utrecht
The Netherlands

Separating the Classes of Recursively Enumerable Languages Based on Machine Size*

Jan van Leeuwen[1] Jiří Wiedermann[2]

[1] Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, the Netherlands
J.vanLeeuwen1@uu.nl

[2] Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. Many years ago it was observed that the r.e. languages form an infinite proper hierarchy $RE_1 \subset RE_2 \subset \cdots$ based on the size of the Turing machines that accept them. Aside from some basic facts, little seems known about it in general. We examine the position of the finite languages and their complements in the hierarchy. We show that for every finite language L one has $L, \bar{L} \in RE_n$ for some $n \leq p \cdot (m - \lfloor \log_2 p \rfloor + 1) + 1$ where m is the length of the longest word in L, c is the cardinality of L, and $p = \min(c, 2^{m-1})$. If $L \in RE_n$, then $\bar{L} \in RE_s$ for some $s = O(n + m)$. We also prove that for every n, there is a finite language L_n with $m = O(n \log_2 n)$ such that $L_n \notin RE_n$ but $L_n, \bar{L}_n \in RE_s$ for some $s = O(n \log_2 n)$. Extending this, we show that there exist families $\{F_n\}_{n \geq 1}$ of finite languages such that $F_1 \subset F_2 \subset \cdots$ where for every n, $F_n \notin RE_n$ but $F_n \in RE_s$ for an s with $s = O(n \log_2 n)$. The proofs make use of several auxiliary results for Turing machines with advice over a fixed alphabet.

1 Introduction

The recursively enumerable languages have a core position in computability theory. The computational complexity of these languages is normally studied in terms of the resources used by the standard Turing machines that accept them, notably time and space [9]. In the late nineteen sixties, Blum [3] suggested to study the effects of program size as well. Schmitt [16] made this idea concrete by proposing the *number of states* of the accepting one-tape Turing machines over a fixed alphabet as a resource.

For $n \geq 1$, let RE_n be the class of recursively enumerable languages acceptable by one-tape Turing machines over a fixed alphabet Σ having at most n states. The primary characteristic of any *size measure* is that there should be only finitely many different machines of any given size. If all other parameters of a Turing machine are fixed (independently of input size), then the 'number

* Version dated March 31, 2014. This research was partially supported by RVO 67985807 and GA ČR grant No. P202/10/1333.

of states' of a machine indeed satisfies this requirement. It easily follows that each RE_n is finite and thus, that the classes form an infinite hierarchy: the *RE-hierarchy*.

Schmitt [16] proved that all classes in the RE-hierarchy are non-empty and that the hierarchy is proper: $RE_1 \subset RE_2 \subset \cdots$. He also proved that for $n \geq 2$, every RE_n contains a finite language, even a finite language that is not in RE_{n-1}. He showed that other well-known families of formal languages, like the non-finite regular languages and the non-regular context-free languages, spread out over successive levels in the hierarchy in a similar way. Little seems known beyond these basic facts, aside from the general properties of size measures [3, 7, 14].

In this paper we examine the position of the finite languages and their complements in the hierarchy more closely. We show e.g. that for every finite language L one has $L, \bar{L} \in RE_n$ for some $n \leq p\cdot(m-\lfloor \log_2 p \rfloor +1)+1$ where m is the length of the longest word in L, c is the cardinality of L and $p = \min(c, 2^{m-1})$. We prove several further detailed results using the same parameters. If $L \in RE_n$, then the n-state Turing machine accepting L may not be always-halting. We show nevertheless that, if $L \in RE_n$, then $\bar{L} \in RE_s$ for some $s = O(n+m)$. Finally, we prove the following main result on the occurrence of the finite languages.

Theorem A *For each $n \geq 1$ there is a finite language L_n with words of size at most $O(n \log_2 n)$ such that $L_n \notin RE_n$ but L_n (and thus also $\bar{L}_n$) $\in RE_s$, for some $s = O(n \log_2 n)$.*

As a corollary we show that there exist families $\{F_n\}_{n\geq 1}$ of finite languages for which $F_1 \subset F_2 \subset \cdots$ and such that for every n, $F_n \notin RE_n$ but $F_n \in RE_s$ for an s with $s = O(n \log_2 n)$.

In the proof of Theorem A we will make use of some results for *Turing machines with advice*. These machines are a special variant of oracle machines in which the oracle is restricted to be a function of the *input size* only rather than of the full input string [10, 1]. In Section 2 we define the basic machine model and recall some useful facts for it. In Section 3 we prove some first results for classifying finite languages depending on the size of their longest string and their cardinality. In Section 4 we consider the complexity of the co-finite languages and in Section 5 we prove Theorem A. In Section 6 we discuss the merit of our results in the context of formal language theory.

We note that, for any language L, the minimum number of states of a Turing machine accepting L corresponds closely to the *descriptional complexity* of L. If $L \in RE_n$, then L is accepted by a Turing machine with a programa that can be represented by a string of length $O(n \log_2 n)$, obtained by encoding the instructions for all state-symbol pairs in binary. Conversely, if L is accepted by a Turing machine with program size r, then $L \in RE_n$ for some $n = O(r/\log_2 r)$. Descriptional complexity has been extensively studied for many classes of special acceptors, notably for finite automata (see [12] for a survey). For the latter, the *state complexity* of finite languages was shown to be $\theta(\frac{2^m}{m})$-bounded, with m as above (cf. [5, 4, 6]). In Section 3 we make this bound more precise, and we show how to improve on it for restricted classes of finite languages. Our

results hopefully add further perspective to the study of the state complexity of languages in general.

2 Preliminaries

We first give the Turing machine-related concepts and conventions used in this paper. We define Turing machines with advice (TM/A's) and show that a concise TM/A for a language L helps in estimating the position of its finite initial segments in the RE-hierarchy. We also give a simple technique to create a concise always-halting TM/A for L from a given TM/A. The advice mechanism will be used in all later sections. Without loss of generality we fix the input alphabet to $\{0,1\}$, the tape alphabet to $\{0,1,B\}$ (with B acting as the blank symbol) and the advice alphabet to $\{0,1\}$.

2.1 Turing machines

The Turing machines we consider will be regular deterministic, one-tape machines with two distinguished halting states: **accept**, and **reject**. Turing machine M is said to *accepts* input x if and only if its computation on x ends in finitely many steps in the state **accept**. If the computation ends in state **reject**, we say that M *rejects* x. Of course the computation may continue *indefinitely*, in which case M is said to *reject* x as well. We assume w.l.o.g. that M never gets stuck in a non-halting state. A machine which either accepts or rejects each of its inputs in finitely many steps, is called an *always-halting machine*.

Although we stay close to the one-tape convention of [16], it will occasionally be easier to use a multi-tape Turing machine in stead. This type of machine has a read-only input tape and one or more separate work tapes. It is well-known that these machines are not more powerful than the one-tape model, but this is usually shown by encoding tracks in alphabet symbols (see e.g. [9], Theorem 6.2). This can not be done here, given the strict fixation of alphabets which we have to respect. The following fact is folklore but a proof seems not widely known. We sketch it below for completeness.

Lemma 1. *Let L be accepted by a q-state multi-tape Turing machine. Then $L \in RE_n$ for $n = O(q)$.*

Proof. Let L be accepted by a q-state multi-tape Turing machine M with t work tapes. We design a one-tape Turing machine M' as follows.

To stay within the conventions of the one-tape model over $\{0,1,B\}$, we encode the contents stored on the various tapes of M symbol-wise in term of fixed equal-size blocks of the form $\mid b < tape >< symbol > \mid$, where b is a bit denoting whether the symbol is currently scanned by the reading head on the tape, $< tape >$ is a fixed-length binary number between 0 and t indicating the tape it is stored on (using 0 for the input tape), and $< symbol >$ the actual symbol stored in the tape-cell. In order to faithfully represents the contents of every particular tape, we only require that the blocks with a same $< tape >$-value appear

in the correct sequential order on the tape and that for each tape precisely one symbol has the b-bit on (to mark the position of the reading head on that tape). Note that the blocks can all be assumed to be of (equal) size u with $u \approx 2 + \log t$, although this is not essential. (The 'vertical bars' are not part of the encoding but only used to denote a block.)

When simulating the actions of M, machine M' can easily skip over the encoded symbols of different tapes, to act on the consecutive symbols of one particular tape. In order to do the simulation, the instructions of M have to be recoded into instruction for M' so that it respects and maintains the representation. We do this as follows. First, for each state s of M, M' will have a number of states of the form

$$< s, goaltape, scanning, sym_0, sym_1, \cdots, sym_t, z >$$

Here s is the state for which M' wants to simulate a move, $< goaltape >$ is the number of the tape which it wants to read, $< scanning >$ records the scanning of the tape number in a block, $sym_0, \cdots sym_t$ are the symbols scanned on the respective tape (insofar as known), and z is one of $O(t)$ 'control states'.

A 'move' of M' typically begins by 'reading' (collecting) the symbols scanned on the tapes. It is done by letting $< goaltape >$ step from 0 to t and for each $< goaltape >$ value move over the tape, stopping only at those blocks which have their b-bit turned on, building up the $< tape >$-value of the scanned block in $< scanning >$ bit by bit and checking whether it matches $< goaltape >$. If so, the scanned $< symbol >$ is stored in the corresponding sym-part of the state. This process is repeated until all tapes have been scanned. After the scanning is complete the corresponding move (i.e. instruction) of M is simulated, by changing the symbols in the scanned blocks or simulating a move to the left or to the right on a tape, by exchanging the b-bit between a block and the 'first' block to the left or to the right that has the same $< tape >$-value, respectively. In z we keep track of the various stages in the process.

It easily follows that the total number of states needed in this simulation is bounded by $q \cdot poly(t) \cdot 2^{t+1} = O(q)$, where $poly(t)$ is a polynomial only depending on t. Only one final detail is needed, namely the transformation of a given input $Bx_1 \cdots x_l B$ to the correct initial representation of the tapes of M. This can be done by a series of moves, first deleting x_1 and moving to the right end of the tape to deposit a block $|10 \cdots 0x_1|$ there, and repeating this for each x_j for j from 2 to l but now printing blocks of the form $|00 \cdots 0x_j|$, and wrapping up by adding a t final blocks of the form $|1 < tape > B|$ to the left end. The tape now looks like

$$\cdots B|1 < 0 > x_1|1 < 0 > x_2| \cdots |1 < 0 > x_l|1 < 1 > B| \cdots |1 < t > B|B \cdots$$

and M' can start. This initial phase requires an additional number of states depending on t only, which is $O(1)$. Machine M' clearly accepts L. □

2.2 Turing machines with Advice

Turing machines with advice are akin to oracle machines as already introduced by Turing [17]. Effectively, an oracle allows inserting outside information into the computation. This information may depend on the concrete input and is given for free to the respective oracle machine whenever the oracle is queried. *Advice* is a special kind of oracle, namely one that returns values which depend only on the size of the input.

Turing machines with advice (TM/A) were introduced by Karp and Lipton [10]. (The idea of 'advice' can already be recognized in [2].) With advice, a Turing machine may, and in general will, easily gain super-Turing computing power, as the advice is not required to be computable.

Definition 1. *An* advice (function) *is a function* $f : \mathbb{N} \to \{0,1\}^*$. *An advice* f *is called* $g(n)$*-bounded for some* $g : \mathbb{N} \to \mathbb{N}$ *if for all* n, $|f(n)| \leq g(n)$.

Technically, a TM/A with advice function f operates on an input of size n in much the same way as a standard Turing machine. However, the machine can also call its advice by entering into a special *query state*. After doing so, the value of $f(n)$ will appear on a special read-only advice tape. From this moment onward, the machine can use the contents of this tape in its computation. (Note that in our set-up, advices are coded over the fixed alphabet $\{0,1\}$.)

One easily verifies that a language L is accepted by a Turing machine M with oracle $\mathcal{O}$ if and only if L is accepted by a TM/A M' with 2^n-bounded advice f. (For the 'if'-part, let $f(n)$ be the bit string of length 2^n in which the i-th bit is 0 or 1 depending on whether the lexicographically i-th word of $\{0,1\}^n$ is accepted by M with its oracle or not. The 'only-if' part follows by definition.)

The following lemma illustrates the power of advice and will be used later on. Given an arbitrary language L, let $L_{\leq m}$ denote the (finite) subset of L consisting of all its words of length at most m. Let $w \in \{0,1\}^*$ be an arbitrary word.

Lemma 2. *Let* L *be accepted by a* q*-state TM/A with advice function* f *such that* $f(k) = w$ *for* $k \leq m$ *and* $f(k)$ *arbitrary otherwise. Then* $L_{\leq m} \in RE_n$ *with* $n = m + |w| + O(q)$.

Proof. Let L be accepted by a TM/A M as described. Design a Turing machine M' as follows. (For clarity we do not aim at an optimal construction.)

Let the input have the following form, with the read-head initially positioned on the leftmost symbol x_1:

$$Bx_1 \cdots x_k B$$

Reading the input M' first goes through $m+1$ states $q_1, \cdots, q_{m+1}$ to check the input's length. If M' reaches the end of the input in state q_i for some $i \leq m$, then it moves from q_i to state w_1 as described below. If M' reaches q_{m+1} exactly when or before the end of the input is reached, then M' moves to a sink state c in which it simply cycles to the end of the input and rejects (as the input was found to have length greater than m).

If M' transfers from a q_i $(1 \leq i \leq m)$ to state w_1 (being at the end of the input), it skips over three blanks at the end of the input and subsequently enters a row of h states $w_1, \cdots, w_h$, with $h = |w|$ and each state corresponding to a symbol of the advice string w. (Note that w is the correct advice for the current input). M' moves from w_1 to w_2 while printing the block $01w_1$ and continues to move from w_j to w_{j+1} $(2 \leq j \leq h-1)$ while printing a block $00w_j$ on the tape. For $j = h$ this is done while moving from w_m to a rewind state d. Arriving in the latter state, M' rewinds its head to the left end of the input, which is recognized by hitting the first blank symbol after passing the block of three.

Using $O(1)$ states, M' now goes through a cycle of deleting the input symbols $x_1, \cdots, x_k$ one at a time on the left and copying them to the right end side of the tape as blocks as shown below. After the last input symbol has been copied, M' moves its reading head to block $11x_1$. The tape now looks as follows:

$$\cdots BBB|01w_1|00w_2|\cdots|00w_h|11x_1|10x_2|\cdots|10x_l|BBB\cdots$$

In this representation the advice and input tape, and the reading heads on these tapes, are represented 'in-line' again and M' can start to simulate the moves of M in the same way as in the proof of Lemma 1. Of course the advice information can only come into play when the simulation of M calls it.

It is easy to modify the transitions of M so they operate on and maintain the in-line encoding. It requires that for each state-symbol pair, M' goes through a small gadget that does the cycling to the left or to the right to pick up the symbols from the tape cells that M currently scans, and deposit changes corresponding to the write and head moves before being ready for a next simulated transition. The gadgets need to be of size $O(1)$ only, which means that we only need a total of $O(q)$ states for it. The result will be that M' maintains a tape of the form

$$B|10\sigma_1|\cdots 10\sigma_r|00w_1|\cdots|01w_j|\cdots|00w_h|10\sigma_{r+1}|\cdots|11\sigma_i|\cdots|10\sigma_v|B$$

where $\sigma_1 \cdots \sigma_r \sigma_{r+1} \cdots \sigma_i \cdots \sigma_v$ is the 'current' contents of the main tape of M and the reading heads are scanning the i'th symbol of this tape (for some $1 \leq i \leq v$) and the j'th one of the advice. Note that extensions of the main tape to the left are written in blocks to the left of the advice. M' clearly accepts $L_{\leq m}$. □

Lemma 2 suggests that one may wish to encode key information for L in a compact advice and have a 'small' TM/A do the rest. Intuitively, the more complex a language is, the longer the advice needed for it. Karp and Lipton [10] indeed stated, without proof, that TM/A's with $g(k)$-bounded advice are more powerful than TM/A's with $h(k)$-bounded advice, provided $g(k) > h(k)$ infinitely often. For $h(k) = o(g(k))$ this was proved by Hermo and Mayordomo [8] using Kolmogorov complexity, the general case was proved by Verbaan [20].

3 Bounding State Complexity

We pursue the position of the finite languages and their complements in the RE-hierarchy. In this Section we give some first, auxiliary bounds on the state complexity of finite languages, using both the length of their longest word and their cardinality as the key parameters.

Notation 1 *For finite $L \subseteq \{0,1\}^\star$ we let $m = \max(L) = \max\{|w| \; | w \in L\}$ and $c = \texttt{card}(L)$, the number of words in L.*

3.1 Bounds from Finite Automata

We can already get interesting bounds by just considering the number of states we need when we only use a finite-state part of a Turing machine i.e. a finite automaton as acceptor, over the fixed input alphabet $\{0,1\}$. We begin with some basic facts.

According to Champarnaud and Pin [5], the question to determine the maximum number of states of any minimal (deterministic) finite automaton recognizing a finite language $L \in \{0,1\}^m$ was apparently first raised by H. Straubing (cf. [5]). They proved that this number is $\theta(\frac{2^m}{m})$, which easily extends to the recognition of all languages with maximum word size m (cf. [4]). Thus, for all finite languages $L \in \{0,1\}^*$ we have $L \in RE_n$, for some $n = \theta(\frac{2^m}{m})$. Gruber and Holzer [6] proved that the latter holds for 'almost all' finite languages with maximum word size m, in a suitable probabilistic model.[3]

We aim at bounds that not only depend on $m = \max(L)$ but also on $c = \texttt{card}(L)$. As a starting point we take the following simple fact.

Lemma 3. *Let L be a finite language with $\texttt{card}(L) \geq 2$. Then $L, \bar{L} \in RE_n$, for some $n \leq c \cdot m$.*

Proof. Let T be the full binary tree of depth $m-1$, with left branches labeled 0 and right branches labeled 1. Let the nodes be initially unlabeled. We add two more states: A and R and label them as 'accept' and 'reject' states already. Note that $m \geq 1$, because $\texttt{card}(L) \geq 2$. We now create a finite automaton $\mathcal{A}$ as follows:

- *I*: enter every word $w \in L$ with $|w| \leq m-1$ into T: start at the root, follow the path as if the nodes are states and w the input, and mark the node of T reached this way as an 'accept' state.
- *II*: now enter every word $w\sigma \in L$ with $|w| = m-1$ and $\sigma \in \{0,1\}$ into T: start at the root, follow the path as if the nodes are states and w the input. Let x be the node of T reached at the frontier of T by following w. Then add an edge labeled σ from x to A.
- *III*: mark every node of T that is not marked as an accept state but is 'on the way' to an accept state (i.e. has an accept node as one its descendants, i.e. including node A), as a 'reject' state.

[3] The results are reminiscent to the Shannon-Lyapunov bounds on circuit-size for Boolean functions in n-variables, see e.g. [21].

- *IV*: ('pruning') delete all nodes of T that did not receive a label, and also delete the edges that lead to them.
- *V*: for every remaining (i.e. *labeled*) node $x \in T \cup \{A, R\}$ and every $\sigma \in \{0, 1\}$, if x lacks or lost an outgoing edge labeled σ then add a σ-labeled edge from x to R.

Note that, by definition, state A must be reached by at least one word of L.

It is easily seen that the automaton $\mathcal{A}$ so constructed is completely specified and accepts precisely the words of L. Moreover, by switching accept and reject labels one obtains an automaton that precisely accepts $\bar{L}$. It remains to estimate the number of states, i.e. the number of nodes that got labeled.

Note that every labeled node of $T \cup \{A\}$ is accounted for as being on the path from the root to an accept node. The total number of these nodes is thus certainly bounded by $c \cdot m$. As $c \geq 2$, the root of T is counted at least twice in this bound. Now counting the root one time less but including R in stead, shows that the total number of states of A is bounded by $c \cdot m$. □

Considering the proof of Lemma 3 it is clear that the given bound is not the sharpest possible. We will improve on the $c \cdot m$-bound in a few steps, first by sharpening the analysis in Lemma 3 in general and then specializing it to classes of languages that satisfy some further constraints.

3.2 Improved bounds

Considering the state-diagram of $\mathcal{A}$, we see that the paths from the root to the accept nodes are likely to overlap considerably, especially in the top part of the tree. A better accounting in the tree leads to the following improved bound.

Theorem 1. *Let L be a finite language with* $\texttt{card}(L) \geq 2$. *Let* $p = \min(c, 2^{m-1})$. *Then* $L, \bar{L} \in RE_n$, *for some* $n \leq p \cdot (m - \lfloor \log_2 p \rfloor + 1) + 1$.

Proof. By assumption $c \geq 2$ and thus $m \geq 1$. We proceed as above and first construct the finite automaton $\mathcal{A}$ as in the proof of lemma 3. (We use the same notation and terminology as in this proof.)

As before the labeled nodes of T can all be accounted for by the nodes that are either on a path from the root to a deepest accepting node in T (i.e. an accepting node with no accepting descendants in T anymore), or on a path from the root to a rejecting node in the frontier of T at depth $m-1$ which is connected to A. This adds up to a total of at most $p = \min(c, 2^{m-1})$ paths. (Note that each deepest accepting node of T can be charged to a depth-$(m-1)$ node among its pruned-away descendents, if the node isn't at depth $m-1$ itself.)

Choose k with $0 \leq k \leq m-1$. We estimate the number of states in $\mathcal{A}$ by the number of labeled nodes in the top part of T of depth k, plus the number of labeled nodes remaining on the paths that stick out of it and lead further down, plus A and R. This amounts to a worst-case bound of

$$(2^{k+1} - 1) + p \cdot ((m-1) - k) + 2 = 2^{k+1} + p \cdot (m - 1 - k) + 1$$

Choosing integer k such that $k = \lfloor \log_2 p \rfloor$, we have $2^{k+1} \leq 2p$ and we get a bound of $p \cdot (m - \lfloor \log_2 p \rfloor + 1) + 1$ on the total number of states. □

Corollary 1. *Let L be a finite language with $2 \leq \mathtt{card}(L) < 2^m$. Then $L, \bar{L} \in RE_n$, for some $n \leq c \cdot (m - \lfloor \log_2 c \rfloor + 1) + 1$.*

Proof. Estimate p by c in the preceding proof and take $k = \lfloor \log_2 c \rfloor$ $(\leq m - 1)$. The bound follows. □

In Corollary 1 the value of c can be restricted to a count of those words of L whose length is greater than $k = \lfloor log_2 c \rfloor$.

With the $\theta(\frac{2^m}{m})$-bound in mind for finite languages in general, we note that Theorem 1 gives the following result for size-bounded languages.

Corollary 2. *Let L be a finite language with $2 \leq c \leq \frac{2^{m+1}}{h(m) \log_2 h(m)}$, for some function h with $h(m) > 1$. Then $L, \bar{L} \in RE_n$ for some n with $n = O(\frac{2^m}{h(m)})$.*

Proof. By assumption we have $\frac{2^{m+1}}{h(m) \log_2 h(m)} \geq 2$, hence $h(m) < 2^m$. Now choose $k = m - \lfloor \log_2 h(m) \rfloor - 1$ in the previous proof. This gives a bound on the number of states of

$$\frac{2^m}{2^{\lfloor \log_2 h(m) \rfloor}} + \frac{2^{m+1}}{h(m) \log_2 h(m)} \cdot \lfloor \log_2 h(m) \rfloor + 1 \leq 2 \cdot \frac{2^{m+1}}{h(m)} + 1$$

which gives the result as claimed. □

3.3 Combining Subtrees

We now improve on the given bounds further, by considering the state diagram of automaton $\mathcal{A}$ in more detail. Let c_j $(0 \leq m)$ be the number of labeled states in level j of $\mathcal{A}$. We are interested in the 'fan-out rate' of the state diagram which, by construction, relates to the fan-out of the accepted words as they are entered into T. The ultimate aim is a further refinement of the $\theta(\frac{2^m}{m})$-bound for general finite languages.

Definition 2. *An* enveloping subset *of the j-th level of T is any subset S_j of the nodes in the corresponding level of the* non-pruned *version of T which contains at least all labeled nodes in this level. Let $s_j = |S_j|$ $(0 \leq j \leq m - 1)$.*

Definition 3. *A finite language L is called $\langle \gamma, R \rangle$-constraint for some constant $\gamma > 0$ and function $R = R(m, c)$, if its corresponding automaton admits enveloping sets S_j such that $s_j \leq O(\frac{R}{2^{\gamma(m-j)}})$ $(0 \leq j \leq m - 1)$.*

Note that *all finite languages are trivially $\langle 1, 2^m \rangle$-constraint.* We will take a general approach and consider languages L which are $\langle \gamma, R \rangle$-constraint for any, possibly tighter, enveloping bound R. In the remainder we assume w.l.o.g. that $m \geq 2$ and that $R \geq 16$.

Given a finite language L, we first modify its automaton $\mathcal{A}$. Consider the construction in Lemma 3. Choose an integer z with $0 \leq z \leq m - 2$ and consider the labeled nodes of T in level $m - z - 1$ (reached by an edge from some labeled

node in level $m-z-2$). We can economize by merging 'identical', i.e. equally labeled subtrees at this level into one, redirecting all transitions from level $m-z-2$ to a single instance of each subtree accordingly. The resulting automaton $\mathcal{A}'$ is clearly equivalent to $\mathcal{A}$ but may have fewer states.

Theorem 2. *Let L be a finite language with $\mathtt{card}(L) \geq 2$. If L is $\langle \gamma, R\rangle$-constraint, then $L, \bar{L} \in RE_n$ for some n with $n = O(\frac{R}{(\log_2 R)^\gamma})$.*

Proof. The labeled nodes of $\mathcal{A}'$ in the first $m-z-2$ levels can be counted by means of the enveloping sets S_j for j up to $m-z-2$. This gives a bound in the order of

$$\Sigma_0^{m-z-2} s_j \leq \Sigma_0^{m-z-2} \frac{R}{2^{\gamma(m-j)}} < \frac{1}{2^\gamma - 1} \cdot \frac{R}{2^{\gamma(z+1)}} = O(\frac{R}{2^{\gamma(z+1)}})$$

To estimate the number of states in the merged subtrees at level $m-z-1$, note that these subtrees all result from complete trees of z levels which got labeled and from which the unlabeled nodes got subsequently removed. Note that the complete trees have $2^{z+1}-1$ nodes and can get labeled in at most $3^{2^{z+1}}$ ways (including 'no label' as a possibility). This bounds the number of different subtrees that can result in the state diagram of $\mathcal{A}$, even if only in a rough way. It follows that the number of states of $\mathcal{A}'$ is bounded in the order of

$$\frac{R}{2^{\gamma(z+1)}} + (2^{z+1} - 1) \cdot 3^{2^{z+1}} + 2$$

Choose $z = \lfloor \log_2 \log_2 R \rfloor - 2$. Then $\frac{1}{4} \log_2 R \leq 2^{z+1} \leq \frac{1}{2} \log_2 R$ and thus we obtain, by substituting:

$$3^{2^{z+1}} \leq 3^{\frac{1}{2} \frac{1}{\log_3 2} \log_3 R} = R^{\frac{1}{\log_3 4}}$$

Substituting further, it follows that

$$\frac{R}{2^{\gamma(z+1)}} + (2^{z+1}-1) \cdot 3^{2^{z+1}} + 2 \leq 4^\gamma \frac{R}{(\log_2 R)^\gamma} + \frac{1}{2} \log_2 R \cdot R^{\frac{1}{\log_3 4}} + 2 = O(\frac{R}{(\log_2 R)^\gamma})$$

which proves the desired bound. □

Theorem 2 generalizes the bound of $O(\frac{2^m}{m})$ states on the complexity of finite languages. This follows because every finite language is $\langle 1, 2^m\rangle$-constraint and the bound follows by mere substitution. For large classes of non-sparse finite languages, Theorem 2 can give better bounds and beat the *cm*-bound as well, as shown below.

Definition 4. *A finite language L is called γ-expansive for some constant γ with $0 < \gamma \leq 1$ if its corresponding automaton admits enveloping sets S_j such that $s_j \leq O(2^{\gamma j})$ $(0 \leq j \leq m-1)$.*

For a γ-expansive language, the enveloping sets can expand only by a factor of at most 2^γ. Clearly a γ-expansive language can have any size up to $1 + \cdots 2^{\gamma(m-1)} + 2 \cdot 2^{\gamma(m-1)} = O(2^{\gamma m})$. Note that *every finite language is 1-expansive.*

Corollary 3. *Let L be a γ-expansive language with cardinality $c = \Omega(\frac{2^{\gamma m}}{m})$, for some γ with $0 < \gamma \leq 1$. Then $L, \bar{L} \in RE_n$ for an n with $n = O(\frac{cm}{(\log_2 cm)^\gamma})$.*

Proof. Let L be a finite language as given, and let $c \geq \alpha \frac{2^{\gamma m}}{m}$. Assume w.l.o.g. that $cm \geq 16$. We will prove that language L is $\langle \gamma, cm \rangle$-constraint.

To show this, rewrite the condition on c as follows, for every j with $0 \leq j \leq m-1$:

$$c \geq \alpha \frac{2^{\gamma m}}{m} \Rightarrow 2^{\gamma j} \leq \frac{1}{\alpha} \frac{cm}{2^{\gamma(m-j)}} \Rightarrow s_j \leq \frac{1}{\alpha} \frac{cm}{2^{\gamma(m-j)}} \Rightarrow s_j \leq O(\frac{cm}{2^{\gamma(m-j)}})$$

Note that the second step follows from the γ-expansiveness of L. We conclude that the requirements of Definition 3 are satisfied and Theorem 2 applies. The corollary follows by taking $R = cm$. □

3.4 Bounds from Turing machines

We were able to improve on the cm-type bound of Lemma 3 by optimizing the automaton $\mathcal{A}$ which we constructed for a finite language L. We may expect to improve on these bounds further when the full power of Turing machines is used. We show that in some cases better bounds can indeed be obtained.

Let L be a finite language with $m = \max(L) \geq 2$ and $c = \texttt{card}(L) \geq 1$. Inspired by Lemma 2 we aim to design a small TM/A for L using as compact an advice as possible.

Let $b = b_L \in \{0,1\}^{2^{m+1}}$ be the *characteristic bit string* of L. This is the bit string consisting of the consecutive values of the characteristic function of L, listed from the empty string up to the strings of length m. (Note that this indeed gives a bit string b with $|b| = 2^{m+1}$.)

Definition 5. *L is said to be* k-blocked *if k is the smallest integer such that the characteristic bit string b consists of at most k blocks of consecutive zeros and at most k blocks of consecutive ones.*

Without loss of generality we may assume that the blocks in b are all maximal and that the zero- and one-blocks alternate. Observe that every finite language is k-blocked for some $k \leq c + 1$.

The following simple observation will be the key for improving over the cm-type bound from Subsections 3.1 and 3.2 again, for large classes of finite languages.

Theorem 3. *Let L be a finite language. If L is k-blocked, then $L, \bar{L} \in RE_n$ with $n \leq O(mk \log_2 \frac{2c}{k})$.*

Proof. Let L be a finite language as specified. We design a TM/A $\mathcal{M}$ for L as follows. We begin by constructing its advice. Of course $b = b_L$ seems perfect for this. With b as advice, only $O(1)$ states would suffice to recognize the words of L. We first show that b can be compressed.

We compress b to a string b' as follows. By assumption b consist of at most k blocks of zeroes and at most k blocks of ones, in alternating order. Say the blocks have lengths $p_1, \cdots, p_k$ and $q_1, \cdots, q_k$ respectively. (Note that $k \leq c$.) Compress each block of, say, p zeros to the binary number for p, and each block of q ones to the binary number for q. It leads to compressed blocks of a total length at most $\log_2 p_1 + \cdots \log_2 p_k + \log_2 q_1 + \cdots + \log_2 q_k + 2k$. Now observing that $q_1 + \cdots + q_k = c$ and thus $p_1 + \cdots + p_k = 2^{m+1} - c$ and using Jensen's inequality, this length is bounded by

$$k \log_2 \frac{2^{m+1} - c}{k} + k \log_2 \frac{c}{k} + 2k = k \log_2 \frac{2^{m+1} c - c^2}{k^2} + 2k = O(mk \log_2 \frac{c}{k})$$

To combine the compressed blocks into one new advice string b', we use a simple in-line encoding, putting the compressed blocks in the correct sequence and pairing every symbol of a compressed zero-block with a 0 and every symbol of a compressed one-block with a 1. Thus b' simply encodes b, with $|b'| = O(mk \log_2 \frac{c}{k})$.

Define the advice function f' by $f(k) = b'$ for $k \leq m$ and $f(k) = \lambda$ (the empty string) for $k > m$. Design a TM/A M that accepts L using f as follows. Given an input x, machine M immediately asks for advice. If the advice is λ, M rejects because the input will be longer than m. If the advice is b', then M goes through a simple routine to decode b' and decompress the blocks to retrieve b. It then reads its input, and inspects the position corresponding to x. If the bit is 1 it accepts x, otherwise it rejects. One easily verifies that M needs only $O(1)$ states for this.

Now apply Lemma 2. It follows that $L = L_{\leq m} \in RE_n$ with n such that $n = m + |b'| + O(1) = O(km \log_2 \frac{2c}{k})$. □

As every finite language is k-blocked for $k \approx c$, Theorem 3 gives a worst-case state complexity that is no better than $O(cm)$. However, as soon as some degree of clustering among the words of L arises, one gets an improvement of this bound. For example, Theorem 3 immediately gives the following.

Corollary 4. *Let L be a finite language. Let L be k-blocked for some $k \leq \frac{(cm)^\gamma}{m}$ with $0 < \gamma \leq 1$. Then $L, \bar{L} \in RE_n$ with $n \leq O((cm)^\gamma \log_2 cm)$.*

4 Positioning the Cofinite Languages

In this section we will be interested in the state complexity of the *complement* of a (finite) r.e. language L. This immediately lead us to the question whether one can convert an arbitrary Turing machine for L into one with a 'small piece of advice' that is always halting.

We show in fact how one can convert any TM/A into an equivalent TM/A that always halts, thus generalizing Proposition 5 from [18] to the case of arbitrary TM/A's. It can also be seen as a generalisation of *Barzdin's lemma* (see [2], Theorem 1) to TM/A's. We show several consequences of this result, including the observation that for any finite language L, $\bar{L}$ is never far away from L in the RE-hierarchy.

Lemma 4. *Let M_1 be a TM/A with $g(k)$-bounded advice. Then there exists an equivalent TM/A M_2 with $(g(k)+k+1)$-bounded advice that halts on each input.*

Proof. Let f_1 be the $g(k)$-bounded advice function of M_1. We build a new advice function f_2 as follows. For each k, let w_k be an input of length k which is accepted by M_1 with the longest running time of all accepting computations of M_1 on inputs of length k (compare [2], Theorem 1). If M_1 has no accepting computation on an input of length k, then let w_k be an arbitrary string of length k. Define advice function f_2 by: $f_2(k) = f_1(k) \cdot w_k\sigma_k$ where $\sigma_k = 1$ if M_1 has accepting computations on inputs of length k and $\sigma_k = 0$ if it doesn't. By design it is always easy to retrieve $f_1(k)$, σ_k and w_k from $f_2(k)$.

Now design a TM/A M_2 as follows. On input w of length $|w| = m$, M_2 first calls its advice $f_2(m)$ and extracts $f_1(m)$, σ_m and w_m. Then it alternately simulates one step of M_1's computation on w and one step of M_1's computation on w_m if the latter is meaningful. Due to the choice of σ_m and w_m, one of the following conditions must arise first:

- $\sigma_k = 0$: then M_2 halts and rejects its input w (as M_1 does not have accepting computations).
- *the computation on w halts in an accept state of M_1*: then M_2 halts and accepts the input w;
- *the computation on w halts in a reject state of M_1*: then M_2 halts and rejects the input w;
- *the computation on w_m halts*: then we know that the computation of M_1 on w will not halt and hence M_2 rejects its input w.

Note that the latter condition can only arise if $\sigma_k = 0$ hasn't arisen first, i.e. if $\sigma_k = 1$. In this case w_m is indeed a correct indicator for the rejection.

Thus, M_2 accepts the same language as M_1, halts on all inputs, and its advice is $(g(k) + k + 1)$-bounded. □

The bound $g(k)+k+1$ in Lemma 4 can be improved to $g(k)+\log_2 c_L(k)+1$, where $c_L(k)$ is the census function of L (see e.g. [19]). This will give a shorter advice in the case of 'sparse' languages.

Corollary 5. *For any Turing machine M_1 - with or without advice - accepting a language L, there exists a TM/A M_2 that accepts $\bar{L}$ and always halts.*

As a special case it follows that the complement of any r.e. language is accepted by an always halting TM/A. We elaborate on this observation in the following variant of Lemma 2.

Lemma 5. *Let L be accepted by a q-state TM/A with advice function f such that $f(k) = w$ for $k \leq m$ and $f(k)$ arbitrary otherwise. Then $L_{\leq m}$ can be accepted by an n-state Turing machine that always halts, for some $n = O(m + |w| + q)$.*

Proof. Let L be accepted by a TM/A M_1 as described. We design a Turing machine M' for $L_{\leq m}$ that always halts. The construction follows the idea of Lemma 4 and applies it to the proof of Lemma 2.

Following the recipe of Lemma 4, we first modify the advice function f of M_1. As we need to accept $L_{\leq m}$, the advice we need to add to $f(k)$ should use $w_k =$ 'the input of length k which is accepted by M_1 with the longest running time of all accepting computations of M_1 on inputs of length k' (or some default value of length k when M_1 does not accept any words of size k). Using the notation from the proof of Lemma 4, let the new advice f' be defined by $f'(k) = f(k) \cdot w_k \sigma_k$.

The further construction amounts to a simulation of M_1 as in Lemma 4, but formatted as in Lemma 2 so as to stay within the framework of one-tape Turing machines with standard work-tape alphabet $\{0, 1, B\}$. To embed the advice f' into the design of the machine we now also have to hardwire the relevant σ_k-values ($0 \leq k \leq m$) into it, which takes no more than $O(m)$ states. If the length check on an input x of length k succeeds, we immediately check σ_k and reject the input if $\sigma_k = 0$.

The remainder of the construction is easily adapted as well. In stead of two in-line tracks we lay out four, in order to carry out a simultaneous simulation of M_1's computation on input x using advice w and on yardstick w_k (also with advice w and only when $\sigma_k = 1$). To set up the extra simulation, we only need $O(m)$ extra states.

The resulting Turing machine halts on every input and uses no more than $O(m + |w| + q)$ states □

We use Lemma 5 to argue that for finite languages, L and $\bar{L}$ can never be very far apart in the RE-hierarchy. Of course L and $\bar{L}$ belong to the *same* class if the smallest Turing machine accepting L is always halting: one simply swaps the accepting and rejecting states to obtain a Turing machine for $\bar{L}$ with the same number of states! However, even if the accepting machine for L is nót always halting, then the indexes of L and $\bar{L}$ in the hierarchy will differ by an amount at most linear in $\max(L)$.

Theorem 4. *Let L be a finite language with $m = \max(L)$ ($m \geq 1$). If $L \in RE_n$, then $\bar{L} \in RE_s$ for some $s = O(n + m)$.*

Proof. Let M be a Turing machine with n states accepting L. Because L is finite with $\max(L) = m$, we have $L = L_{\leq m}$.

We now construct a Turing machine M' that accepts $L_{\leq m}$ and that always halts, using the technique of Lemma 5. Note that M can be viewed as a TM/A with advice function $f(k) = \lambda$. Hence, the machine M' constructed in the proof of Lemma 5 will have $O(n+m)$ states. By swapping the accepting and rejecting states, we have a machine that accepts $\bar{L}$. Hence $\bar{L} \in RE_s$ for $s = O(n+m)$. □

5 Separating the RE-hierarchy

We now return to the RE-hierarchy. How are the finite languages spreading through the RE-hierarchy concretely? In Section 3 we gave some first results on the position of both L and $\bar{L}$ depending on $\max(L)$ and $\texttt{card}(L)$, but these were all upperbounds. In this Section we give a more fundamental bounding result

for the finite languages and their complements in the RE-hierarchy. We take the finite languages as the core sets.

We now prove the result stated earlier as Theorem A. The proof implicitly uses various techniques developed in Section 2 and in the previous Section.

Theorem 5. *For each $n \geq 1$ there is a finite language L_n with $\max(L_n) = O(n \log_2 n)$ such that $L_n \notin RE_n$ but $L_n \in RE_s$, for some $s = O(n \log_2 n)$.*

Proof. We construct the languages L_n by a diagonalisation argument. We first state our conventions and then define L_n.

Conventions
Every Turing machine of n states can be encoded in $N = N_n = O(n \log_2 n)$ bits, by a fixed standard representation of the transition table over our standard alphabet. The encoding can be done in such a way that it is easily recognized whether a given string of N bits is the encoding of an *actual* Turing machine with n-states. We use N throughout to denote the fixed code size for n-state machines, omitting the subscript n when n is understood.

Language L_n
We now want to associate a Turing machine with *every* string $w \in \{0,1\}^N$. In order to achieve it, we consider a standard e.g. lexicographic enumeration of the strings in $\{0,1,B\}^N$ (with wrap-around) and let each string $w \in \{0,1\}^N$ correspond to the machine described by *the first string following in the enumeration starting from w that corresponds to a concrete Turing machine encoding.* This is well-defined, due to the wrapping around of the enumeration.

Let M_w be the n-state Turing machine thus associated with a string $w \in \{0,1\}^N$. (Note that, as a consequence, different strings w can thus correspond to the same n-state machine but this is no problem.) Define L_n by

$$L_n = \{w \mid w \in \{0,1\}^N, w \text{ is not accepted by } M_w\}$$

By a standard argument one sees that $L_n \notin RE_n$. For, suppose that L_n could be accepted by an n-state Turing machine Z. Let $Z = M_w$ for some N-bit code w. Then the assumptions $w \in L_n$ and $w \notin L_n$ both lead to a contradiction. We will aim to design a Turing machine that *does* accept L_n.

Machine M
Before we do so, we design an auxiliary machine. Define $L = \bigcup_{n \geq 1} L_n$. By the same argument as above one sees that L is not accepted by any Turing machine with finitely many states. Indeed, assuming that L would be accepted by an r-state Turing machine for some r, will give a contradiction in the same way as before. However, by an argument similar to that in Lemma 4 one easily shows that L is accepted by a TM/A M with advice function f defined by

$f(k) =$
a. λ : if there is no r such that $k = N_r$,
b. λ : if there is no string string w with $|w| = k$ for which M_w halts on input w,

c. w : if w is the lexicographically least string with $|w| = k$ for which M_w halts on w using the longest possible number of steps over all w of size k for which M_w halts on w.

On any given input x, with $|x| = k$ for some $k > 0$, M checks whether x is of length $N = N_r$ for some r. If it is not, M rejects x. Otherwise M simply calls its advice $f(k)$. If $f(k) = \lambda$ then M accepts x. If $f(k) = w \neq \lambda$, then M proceeds by interlacing the simulation of M_x on x and that of M_w on w, using the latter as a yardstick to stop the simulation if M_x would run infinitely long on x. Note that M is always halting. Suppose M has b states (b a constant).

Accepting L_n
We now modify M into a TM/A M' that accepts L_n, viewed as a slice of L. The easiest way to do this is to allocate an extra N_n states and let M' check that $|x| = N_n$. One can do this also in $O(n + \log_2 n)$ states and create the needed yardstick of length N_n on the tape, namely by spending $O(n)$ states to lay out a yardstick of size n and iterating through another $O(\log_2 n)$ states $O(n)$ times. This will give a suitable M' for L_n. However, as the advice will now be called *only* on inputs of the right length $k = N_n$, we can restrict the range of the advice f and simplify it to the advice function f' defined by

$f'(k) =$
a. λ : if $k > N_n$,
b. λ : if there is no string w with $|w| \leq N_n$ for which M_w is well-defined (i.e. $|w| = N_r$ for some r) and halts on w, and
c. w : if w is the lexicographically least string with $|w| \leq N_n$ for which M_w is well-defined and halts on w using the longest possible number of steps over all w of length $\leq N_n$ for which M_w is well-defined and halting on w.

It is easily argued that M' with advice f' is a valid TM/A accepting precisely the strings of L_n. Observe that M' uses $q = O(b+N_n) = O(N_n)$ states (or in the more economic version, $O(b + n + \log_2 n) = O(n)$ states). The advice function f' is N_n-bounded. M' is again always halting.

Wrapping up
By applying lemma 5 it now follows that $L_n = (L_n)_{\leq N_n}$ can be accepted by an s-state Turing machine that always halts, for some $s = O(N_n + N_n + N_n) = O(N_n) = O(n \log_2 n)$. □

The proof of Theorem 5 is non-constructive because machine M cannot be found by effective means. Nevertheless, the proof is sufficient for giving upper- and lowerbounds on the position of each language L_n in the RE-hierarchy. Note that by swapping accepting and rejecting states in M' one obtains that $\bar{L}_n$ RE_s as well. It can be noticed that the $O(n \log_2 n)$-bound for s is due entirely to the coding we used.

We can draw a further conclusion from the proof of Theorem 5 which shows even more clearly how the finite languages separate the classes of the RE-hierarchy.

Corollary 6. *There exist a family* $\{F_n\}_{n\geq 1}$ *with* $F_1 \subset F_2 \subset \cdots$ *such that for every* n, $\max(F_n) = O(n \log_2 n)$, $F_n \notin RE_n$, *but* $F_n \in RE_s$ *with* $s = O(n \log_2 n)$.

Proof. Let $\{0,1\}^{*k}$ denote the set of binary strings of length *less than* k. Using the languages L_n as defined in the proof above, let $F_n = L_n \cup \{0,1\}^{*N_n}$. One easily verifies that the arguments given in the proof above for L_n apply to F_n just the same. In particular one sees that $F_n \notin RE_n$.

We now only need to modify M' so it no longer rejects its inputs x with $|x| < N_n$ but accepts them. The advice function f' can remain unchanged as the advice is never called on inputs of length less than N_n.

It follows by the same argument as in the proof of Theorem 5 that $F_n \in RE_s$ for some $s = O(n \log_2 n)$. □

As a curiosity we note that the family $\{F_n\}_{n\geq 1}$ just constructed has the property that $\bigcup_{n\geq 1} F_n = \{0,1\}^*$. In other words, $\{0,1\}^*$ can be decomposed into an infinite ascending chain of finite languages which spread to higher and higher classes in the RE-hierarchy, although $\{0,1\}^* \in RE_1$.

6 Conclusions

The study of program size originated with Blum [3]. The notion was intensively studied in abstract complexity theory in the nineteen seventees and later. Starting with the classic paper of Meyer *et al.* [15], notions of program size and state complexity have been put forward as measures of *descriptional complexity* for functions, sets, automata, and other systems.

In this note we studied the RE-hierarchy, the hierarchy of r.e. languages based on the number of states ('size') of the accepting Turing machine. The basic results for this hierarchy go back more than 40 years as well, to Schmitt [16]. The hierarchy is indicative for the computational richness that can be encoded into a Turing machine's program. In particular it gives a way to classify languages by the number of states needed to accept them. Considerable attention to state complexity was given e.g. for the case of finite automata.

Our main aim has been to add some hopefully new perspectives to the combinatorial nature of the RE-hierarchy. We have investigated the position of the finite languages in the hierarchy, using not only $\max(L)$ but also $\texttt{card}(L)$ as a parameter. We proved several results which vary on or improve the $\theta(\frac{2^m}{m})$-bound for finite languages known from automata theory. However, our main results have aimed to separate the classes of the RE-hierarchy by 'concrete' finite languages.

The separation results proved in Section 5 shown that the classes in the RE-hierarchy can be separated by finite languages F with very limited $\max(F)$ value. It would be interesting to refine the results along this line and achieve tighter bounds. Also, can the results be extended from finite languages to e.g. the non-finite regular languages or non-regular contextfree languages like Schmitt's results? One can probably pad the languages in the proof of Theorem 5 with an infinite part to make them non-finite and have the property one desires. We leave these and other questions to future studies.

As a new technique in this domain we have employed various results for Turing machines 'with advice'. We have shown that optimizing the (length of the) advice in a Turing machine can be an effective intermediate step in bounding the state complexity of finite languages. It would be interesting to explore this technique for other classes of languages as well.

References

1. J.L. Balcázar, J. Díaz, J. Gabarró, *Structural Complexity* I, Second Edition, Springer, 1995.
2. Ja.M. Barzdin′, Complexity of programs to determine whether natural numbers not greater than n belong to a recursively enumerable set, *Soviet Math. Dokl.* 9:5 (1968) 1251-1254.
3. M. Blum, On the size of machines, *Information and Control* 11 (1967) 257-265.
4. C. Câmpeanu, W.H. Ho, The maximum state complexity of finite languages, *J. Automata, Languages, and Combinatorics* 9:2/3 (2004) 189-202.
5. J. Champarnaud, J.-E. Pin, A maxmin problem on finite automata, *Discr. Appl. Math.* 23:1 (1989) 9196.
6. H. Gruber, M. Holzer, On the average state and transition complexity of finite languages, *Theor. Comput. Sci.* 387 (2007) 155-166.
7. J. Hartmanis, J. Hopcroft, An overview of the theory of computational complexity, *J.ACM* 18:3 (1971) 444-475.
8. M. Hermo, E. Mayordomo, A note on polynomial-size circuits with low resource-bounded Kolmogorov complexity, *Math. Systems Theory* 27:4 (1994) 347-356.
9. J.E. Hopcroft, R. Motwani, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd Edition, Addison Wesley, 2006.
10. R.M. Karp, R.J. Lipton, Some connections between non-uniform and uniform complexity classes, *Proc. Twelfth Ann. ACM Symp. on Theory of Computing*, ACM Press, 1980, pp. 302-309.
11. R.M. Karp, R.J. Lipton, Turing machines that take advice, *L'Enseignement Mathématique* II^e Série, Tome XXVIII (1982) 191-209. (Extended version of [10].)
12. M. Kutrib, G. Pighizzini, Recent trends in descriptional complexity of formal languages, *Bulletin European Assoc. for Theoretical Computer Science* No. 111, October 2013, http://www.eatcs.org/beatcs/index.php/beatcs/issue/view/10.
13. M. Li, P. Vitányi, *An introduction to Kolmogorov complexity and its applications*, 3rd ed., Springer, New York, 2008.
14. M. Machtey, P. Young, *An Introduction to the General Theory of Algorithms*, North-Holland, Elsevier, 1978.
15. A.R. Meyer, M.J. Fischer, Economy of expression by automata, grammars, and formal systems, *Proc. 12th Annual IEEE Symposium on Switching and Automata Theory* (SWAT'71), IEEE Computer Society, 1971, pp. 188-191.
16. A.A. Schmitt, The state complexity of Turing machines, *Information and Control* 17 (1970) 217-225.
17. A.M. Turing, Systems of logic based on ordinals, *Proc. London Math. Soc.*, Series 2, 45 (1939) pp. 161-228.
18. J. van Leeuwen, J. Wiedermann, The Turing machine paradigm in contemporary computing, in: B. Engquist and W. Schmidt (Eds), *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, 2001, pp. 1139-1155.

19. J. van Leeuwen, J. Wiedermann, Turing machines with one-sided advice and the acceptance of the co-RE languages, Techn. Report UU-CS-2014-003, Dept of Information and Computing Sciences, Utrecht University, 2014,
20. P.R.A. Verbaan, *The Computational Complexity of Evolving Systems*, Ph.D. Thesis, Faculty of Science, Utrecht University, 2006.
21. H. Vollmer, *Introduction to circuit complexity*, Springer-Verlag, Berlin, 1999.