Comparing Verboseness for Finite Automata and Turing Machines

Till Tantau

Technische Universität Berlin
Fakultät für Elektrotechnik und Informatik
10623 Berlin, Germany
tantau@cs.tu-berlin.de

Abstract. A language is called (m, n)-*verbose* if there exists a Turing machine that enumerates for any n words at most m possibilities for their characteristic string. We compare this notion to (m, n)-*fa-verboseness*, where instead of a Turing machine a finite automaton is used. Using a new *structural* diagonalisation method, where finite automata trick Turing machines, we prove that all (m, n)-verbose languages are (h, k)-verbose, iff all (m, n)-fa-verbose languages are (h, k)-fa-verbose. In other words, Turing machines and finite automata behave in exactly the same way with respect to inclusion of verboseness classes. This identical behaviour implies that the Nonspeedup Theorem also holds for finite automata. As an application of the theoretical framework, we prove a lower bound on the number of bits that need to be communicated to finite automata protocol checkers for nonregular protocols.

1 Introduction

Turing machines and finite automata share some properties that they do not appear to share with computational models 'in between'. While it is well known that both for Turing machines and for finite automata *nondeterminism* is exactly as powerful as *determinism*, for polynomial-time computations this is the famous 'P = NP?' question. In this paper we study a property that Turing machines and finite automata share, but which they provably do not share with classical resource-bounded computational models in between. This property is the *inclusion structure of verboseness classes*.

Verboseness classes for Turing machines were originally defined in an effort to better understand the structure of undecidable problems. Even if a language L is nonrecursive, it might still be possible to compute some partial information about it. A language L is called (m, n)-*verbose* [7] if there exists a Turing machine that does the following: It gets n input words and then does a possibly infinite computation during which it prints some bit strings onto an output tape. One of the bit strings must be the characteristic string of the words with respect to L and at most m different bit strings may be printed. The class of all (m, n)-verbose languages is denoted $\mathrm{V}(m, n)$.

All languages are in $\mathrm{V}(2^n, n)$, and $\mathrm{V}(1, n)$ contains exactly the recursive languages. The structure between these two extremes has been subject to thorough investigation. Beigel's Nonspeedup Theorem [3, 5] states $\mathrm{V}(n, n) = \mathrm{V}(n-1, n) = \cdots = \mathrm{V}(1, n)$. All recursively enumerable and all semirecursive [10] languages are in $\mathrm{V}(n+1, n)$. It is known [7] that $\mathrm{V}(3, 2) = \mathrm{V}(n+1, n)$ for all $n \geq 2$. More generally, in [7] Beigel, Kummer, and Stephan describe a decision procedure for checking whether $\mathrm{V}(m, n) \subseteq \mathrm{V}(h, k)$ holds for given numbers m, n, h, and k.

Verboseness has also been studied extensively for the situation where the enumerating Turing machine is restricted to use only a polynomial amount of time. The inclusion structure of polynomial-time verboseness classes, denoted $\mathrm{V_P}(m, n)$ in the following, is quite different from the structure in the recursive setting. For example $\mathrm{V_P}(m, n) \subsetneq \mathrm{V_P}(m+1, n)$ for all $m < 2^n$. Languages that are in $\mathrm{V_P}(n, n)$ for some n are commonly called *cheatable* [4], languages in the class $\mathrm{V_P}(2^n - 1, n)$ are called *n-approximable* [6] or *n-membership comparable* [15].

The notion of verboseness can also readily be defined for finite automata. The classes $\mathrm{V_{fa}}(m, n)$ are defined via multi-tape automata whose output is specified by the type of the last state reached on a given input. Such an automaton witnesses $L \in \mathrm{V_{fa}}(m, n)$ as follows: When run on n input words, placed on n synchronously read input tapes, the output attached to last state reached by the automaton must be a set of size at most m containing the characteristic string of the words. The classes $\mathrm{V_{fa}}(m, n)$ will be called *fa-verboseness classes*. Similar transfers of recursion or complexity theoretic concepts to finite automata have already been studied in 1976 by Kinber [11] and much more recently by Austinat, Diekert, Hertrampf, and Petersen [1, 2].

As in the recursive setting, all languages are in $\mathrm{V_{fa}}(2^n, n)$ and $\mathrm{V_{fa}}(1, n)$ contains exactly the regular languages. Austinat et al. [1, 2] present different examples of fa-verbose languages that lie between these extremes: For every infinite bit string b both the set of all words that are lexicographically smaller than b and the set of all prefixes of b are in $\mathrm{V_{fa}}(3, 2)$. This class contains context-sensitive languages that are not context-free, and context-free languages that are not regular. However, for all $m < 2^n$ the classes $\mathrm{V_{fa}}(m, n)$ do not contain any infinite context-free languages like $\{\, a^i b^i \mid i \geq 0 \,\}$ that lack infinite regular subsets.

What does the inclusion structure of the classes $\mathrm{V_{fa}}(m, n)$ look like? As the inclusion structure of polynomial-time verboseness classes is quite different from the structure of their recursive counterparts, it might seem surprising that *the inclusion structure of the classes* $\mathrm{V_{fa}}(m, n)$ *is exactly the same as for* $\mathrm{V}(m, n)$. Using the new notion of *structural* diagonalisation, where finite automata trick Turing machines, we can even show that $\mathrm{V_{fa}}(m, n) \setminus \mathrm{V}(h, k)$ is nonempty unless $\mathrm{V}(m, n) \subseteq \mathrm{V}(h, k)$. Thus $\mathrm{V}(m, n) \subseteq \mathrm{V}(h, k)$ iff $\mathrm{V_{fa}}(m, n) \subseteq \mathrm{V_{fa}}(h, k)$ iff $\mathrm{V_{fa}}(m, n) \subseteq \mathrm{V}(h, k)$.

An immediate corollary is that the Nonspeedup Theorem also holds for finite automata. This is interesting in itself. While the Nonspeedup Theorem is an important theoretical result, it has limited practical importance as it concerns the properties of languages that are not even recursive. Opposed to this, we shall see that its finite automata counterpart can be used to prove a lower bound on

the number of bits that need to be communicated to finite automata protocol checkers for nonregular protocols.

This paper is organised as follows. In *Section 2* we review functions that are *enumerable* by Turing machines and transfer this notion to finite automata. We prove a key lemma, which holds both for Turing machines and for finite automata. In *Section 3* we show that the Generalised Nonspeedup Theorem [7] is an easy consequence of the key lemma and thus also holds for finite automata. We then review the combinatorial tools used in [7] to describe the inclusion structure of verboseness classes and apply them to fa-verboseness. In *Section 4* we complete the characterisation of the inclusion structure of fa-verboseness classes by use of a structural diagonalisation argument. Finally, *Section 5* presents a consequence of the Finite Automata Nonspeedup Theorem for the difficulty of protocol checking using finite automata.

2 Recursive and Finite Automata Enumerability

Before focusing on verboseness in the following sections, in this section we study the more general concept of *enumerability* of functions. This notion was first introduced by Cai and Hemaspaandra [8] in the context of polynomial-time computations and only later transfered to recursive computations.

Definition 1 ([14, 9]). *A function f, taking n-tuples of words as input, is in the class* $\mathrm{EN}(m)$ *if there exists a deterministic Turing machine with n input tapes that does the following: Upon input of words $w_1, \ldots, w_n$, placed on the input tapes, it starts a possibly infinite computation during which it prints words onto an output tape. No more than m words may be printed, and one of them must be $f(w_1, \ldots, w_n)$.*

For Turing machines it is not important whether the input words are placed on different tapes or just alongside each other on a single input tape, separated by marker symbols. For finite automata this *does* make a crucial difference. We shall only consider the more interesting situation where multiple tapes are used, see below.

The connection between enumerability and verboseness is discussed in detail in the next section, but note already that a language L is (m, n)-verbose iff $\chi_L^n \in \mathrm{EN}(m)$. Here χ_L^n is the n-fold characteristic function of L, which takes a tuple $(w_1, \ldots, w_n)$ as input and yields a bit string of length n whose i-th position is 1 iff $w_i \in L$.

We now explain how finite automata compute functions. As done in [1, 2] we use deterministic automata that instead of just accepting or rejecting a word may produce many different outputs, depending on the type of the last state reached. Formally, instead of a set of accepting states the automata are equipped with an output function $\gamma\colon Q \to \Gamma$ that assigns an output $\gamma(q) \in \Gamma$ to each state $q \in Q$. The *output* $\mathrm{out}_A(w)$ of an automaton A on input w is the value $\gamma(q)$ attached to the *last state q reached by A upon input w*. An automaton A *computes a*

function f if $f(w) = \mathrm{out}_A(w)$ for all w. Note that only functions having finite range can be fa-computed in this sense.

As mentioned earlier, to fa-compute a function taking n-tuples of words as input, we use n input tapes. Each word is put onto one tape, shorter words padded with blanks at the end such that all words have the same length. Then the automaton scans the words *synchronously*, meaning that in each step all heads advance exactly one symbol. Equivalently, one may also think of the input words being fed to a single-tape automaton in an interleaved fashion: first the first symbols of all input words, then the second symbols, then the third symbols, and so forth.

Having fixed how finite automata *compute* functions, we can now define how they *enumerate* them.

Definition 2. *A function f, taking n-tuples of words as input, is in the class $\mathrm{EN}_{\mathrm{fa}}(m)$ if there exists a deterministic n-tape automaton A, having sets as output, such that for all words $w_1, \ldots, w_n$ the set $\mathrm{out}_A(w_1, \ldots, w_n)$ has size at most m and contains $f(w_1, \ldots, w_n)$.*

Our first result is the key lemma below. It makes the same claim twice, only the first time for enumerability and the second time for fa-enumerability. In a sense this 'double claim' is the deeper reason why the inclusion structures of verboseness and fa-verboseness classes coincide. The *product* $f \times g$ of two functions f and g used in the lemma is defined in the usual way by $(f \times g)(u, v) := \big(f(u), g(v)\big)$. The first part of the proof is similar to the proof of the Generalised Nonspeedup Theorem given in [7], but see also [16].

Lemma 3 (Key Lemma).
If $f \times g \in \mathrm{EN}(n + m)$, then either $f \in \mathrm{EN}(n)$ or $g \in \mathrm{EN}(m)$. Likewise, if $f \times g \in \mathrm{EN}_{\mathrm{fa}}(n + m)$, then either $f \in \mathrm{EN}_{\mathrm{fa}}(n)$ or $g \in \mathrm{EN}_{\mathrm{fa}}(m)$.

Proof. Assume $f \times g \in \mathrm{EN}(n+m)$ via a machine M. Let us write $M(u, v)$ for the set of pairs enumerated by M upon input of the pair (u, v). Suppose there exists a word u such that for all words v the set $P_{u,v} := \big\{\, y \mid \big(f(u), y\big) \in M(u, v) \,\big\}$ has size at most m. Then $g \in \mathrm{EN}(m)$ and we are done. So suppose no such word u exists. Then for all words u there exists a word v such that the set $P_{u,v}$ has size *at least* $m+1$. This implies that in $M(u, v)$ at least $m+1$ pairs have the same first component, and hence $\{\, x \mid (x, y) \in M(u, v) \,\}$ has size *at most* $n + m - m = n$. But then $f \in \mathrm{EN}(n)$, as upon input of a word u, using dovetailing, we can *search* for a word v with the property, that in $M(u, v)$ at least $m + 1$ pairs have the same first component. Having found this word, we stop the search and enumerate $\{\, x \mid (x, y) \in M(u, v) \,\}$.

Now assume $f \times g \in \mathrm{EN}_{\mathrm{fa}}(n + m)$ via a 2-tape automaton A with output function γ. Once more, assume there exists a word u such that for all words v the set $P_{u,v} := \big\{\, y \mid \big(f(u), y\big) \in \mathrm{out}_A(u, v) \,\big\}$ has size at most m. Then $g \in \mathrm{EN}_{\mathrm{fa}}(m)$ via an automaton B that *simulates* what A would do upon input (u, v). Such a simulation is possible as u is a single fixed word. When the simulation finishes, the automaton B will 'know' the state q in which A would have ended up upon

input of (u, v). The automaton B can then output $P_{u,v}$ instead of $\mathrm{out}_A(u, v)$, thus proving $g \in \mathrm{EN}_{\mathrm{fa}}(m)$.

Next assume that for all words u there exists a word v such that the set $P_{u,v}$ has size at least $m + 1$. We then show $f \in \mathrm{EN}_{\mathrm{fa}}(n)$ using a variant of the search trick used in the recursive setting. The aim is to 'search' for a word v such that in $\mathrm{out}_A(u, v)$ at least $m + 1$ pairs have the same first component – we know that such a word exists, we just have to find it. The idea is to first construct a *nondeterministic automaton C with ϵ-moves*. This automaton has just one input tape, on which it finds an input word u. As its first step it branches nondeterministically to all states the automaton A reaches when it reads the first symbol of u on its first tape and some arbitrary symbol on the second tape (including the blank symbol, which corresponds to guessing the end of the word on the second tape). In the second step C branches to the states A reaches upon then reading the second symbol of u plus some arbitrary symbol on the second tape and so on. When the end of u is reached, C may go on simulating A using ϵ-moves: It nondeterministically branches to the states A would reach reading a blank on the first tape and some arbitrary symbol on the second tape. Note that *on some nondeterministic path the automaton C will reach the state reached by A upon input (u, v)*.

We turn C into a deterministic automaton D using the standard power set construction on its states. Upon input of a word u the automaton D will end its computation in a 'power state' p that is exactly the set of all states reached by C upon input u. We define the output attached to p in D as the intersection of all sets $\{ x \mid (x, y) \in \gamma(q) \}$ with $q \in p$, thus ensuring that $f(u)$ is always an element of $\mathrm{out}_D(u)$. Due to the existence of the word v, there must exist a state $q \in p$ such that in the set $\gamma(q)$ at least $m + 1$ pairs have the same first component. Thus the intersection output by D has size at most n. □

The contraposition of the lemma states that if $f \notin \mathrm{EN}(n)$ and $g \notin \mathrm{EN}(m)$, then $f \times g \notin \mathrm{EN}(n + m)$; and likewise for finite automata. This can be seen as a lower bound on the enumerability of the product of two functions: If f and g are not n- and m-enumerable respectively, then $f \times g$ is not $(n + m)$-enumerable. Compare this to the trivial upper bound: If $f \in \mathrm{EN}(n)$ and $g \in \mathrm{EN}(m)$, then $f \times g \in \mathrm{EN}(n \cdot m)$; and likewise for finite automata.

Repeatedly applying the lemma yields that if $f_1 \times \cdots \times f_\ell \in \mathrm{EN}(n_1 + \cdots + n_\ell)$, then $f_i \in \mathrm{EN}(n_i)$ for some i; and likewise for finite automata. In particular, if $f_1 \times \cdots \times f_n$ is in $\mathrm{EN}(n)$, respectively $\mathrm{EN}_{\mathrm{fa}}(n)$, then at least one f_i is recursive, respectively fa-computable.

3 Application of the Key Lemma to Verboseness

In this section we apply the key lemma to verboseness classes. We first introduce verboseness of *functions* as a useful and elegant generalisation of the verboseness of *languages*. In the following, f^n always denotes the n-fold product $f \times \cdots \times f$.

Definition 4. *A function f is* (m, n)-verbose *if* $f^n \in \mathrm{EN}(m)$. *A function f is* (m, n)-fa-verbose *if* $f^n \in \mathrm{EN}_{\mathrm{fa}}(m)$. *A language is* (m, n)-verbose*, respectively*

(m,n)-fa-verbose, *if its characteristic function is. The classes of* (m,n)*-verbose and* (m,n)*-fa-verbose languages are denoted* $\mathrm{V}(m,n)$ *and* $\mathrm{V}_{\mathrm{fa}}(m,n)$.

Theorem 5. *If* f *is* $(m+h,n+k)$*-verbose, then* f *is either* (m,n)*-verbose or* (h,k)*-verbose; and likewise for fa-verboseness.*

Proof. By the key lemma, if $f^{n+k} = f^n \times f^k \in \mathrm{EN}(m+h)$ we have either $f^n \in \mathrm{EN}(m)$ or $f^k \in \mathrm{EN}(h)$; and likewise for finite automata. □

Corollary 6 (Generalised Nonspeedup Theorem). *We have*

$$\mathrm{V}(m+h,n+k) \subseteq \mathrm{V}(m,n) \cup \mathrm{V}(h,k),$$
$$\mathrm{V}_{\mathrm{fa}}(m+h,n+k) \subseteq \mathrm{V}_{\mathrm{fa}}(m,n) \cup \mathrm{V}_{\mathrm{fa}}(h,k).$$

Corollary 6 can be used to prove inclusions of verboseness classes. For example we immediately get $\mathrm{V}(m+1,n+1) \subseteq \mathrm{V}(m,n)$. A systematic study of the derivable inclusions leads to the definition of *goodness* [7]. In the following an *n-pool* is a subset of $\{0,1\}^n$. For a bit string $b \in \{0,1\}^n$ with individual bits $b_1 \cdots b_n = b$ we write $b[i_1,\ldots,i_\ell] := b_{i_1} \cdots b_{i_\ell} \in \{0,1\}^\ell$. For an n-pool P we define $P[i_1,\ldots,i_\ell] := \{\, b[i_1,\ldots,i_\ell] \mid b \in P \,\}$.

Definition 7 ([7]). *For a* k*-pool* P *let* $g_P(n)$ *be the maximum of* $\big|P[i_1,\ldots,i_n]\big|$ *taken over all indices* $i_1,\ldots,i_n \in \{1,\ldots,k\}$. *A pool* P *is* (m,n)-good, *if for every partition* $n_1 + \cdots + n_\ell = n$ *with* $1 \le n_i \le n$ *we have*

$$g_P(n_1) + \cdots + g_P(n_\ell) \le m + \ell - 1.$$

Theorem 8. *Let every* (m,n)*-good* k*-pool have size at most* h. *Then*

$$\mathrm{V}(m,n) \subseteq \mathrm{V}(h,k),$$
$$\mathrm{V}_{\mathrm{fa}}(m,n) \subseteq \mathrm{V}_{\mathrm{fa}}(h,k).$$

Proof. As the claim $\mathrm{V}(m,n) \subseteq \mathrm{V}(h,k)$ is proven as part of Theorem 4.3 in [7], we show only the second claim. We revisit the proof of [7], whose key ingredient is the Generalised Nonspeedup Theorem, which is also correct for finite automata as we have seen in Corollary 6.

Let $L \in \mathrm{V}_{\mathrm{fa}}(m,n)$. To prove $L \in \mathrm{V}_{\mathrm{fa}}(h,k)$, we construct an automaton B that computes for every k words $w_1,\ldots,w_k$ an (m,n)-good k-pool P containing the bit string $\chi^k_L(w_1,\ldots,w_k)$. By assumption, P will have size at most h. For each $i \in \{1,\ldots,n\}$ let m_i be the smallest number such that $L \in \mathrm{V}_{\mathrm{fa}}(m_i,i)$ via some finite automaton A_i. Applying Corollary 6 to $L \in \mathrm{V}_{\mathrm{fa}}(m_{i+j},i+j)$ we get $L \in \mathrm{V}_{\mathrm{fa}}(m_i-1,i) \cup \mathrm{V}_{\mathrm{fa}}(m_{i+j}-m_i+1,j)$. As L is not an element of $\mathrm{V}_{\mathrm{fa}}(m_i-1,i)$ by the choice of m_i, the language L must be in $\mathrm{V}_{\mathrm{fa}}(m_{i+j}-m_i+1,j)$. Because of the minimality of m_j this yields $m_j \le m_{i+j}-m_i+1$ and thus $m_i+m_j \le m_{i+j}+1$.

The automaton B runs the following algorithm: Upon input of the k words it simulates simultaneously for all $j \in \{1,\ldots,n\}$ and for all indices $i_1,\ldots,i_j \in \{1,\ldots,k\}$ the automata A_j on the input words $w_{i_1},\ldots,w_{i_j}$. When B reaches the end of the input words it will 'know' what all the $\mathrm{out}_{A_j}(w_{i_1},\ldots,w_{i_j})$ would be. It

then outputs the largest set of bit strings that is consistent with the output of all the automata A_j on the different selections of words. More precisely, it outputs the largest k-pool P with the property that $P[i_1, \ldots, i_j] \subseteq \mathrm{out}_{A_j}(w_{i_1}, \ldots, w_{i_j})$ for all $i_1, \ldots, i_j \in \{1, \ldots, k\}$ and all $j \in \{1, \ldots, n\}$. Note that P indeed contains $\chi_L^k(w_1, \ldots, w_k)$.

We claim that the pool P output by B is (m, n)-good. Let $n_1 + \cdots + n_\ell = n$ be any partition with $1 \leq n_i \leq n$. From the construction of P and the definition of g_P we directly get $g_P(j) \leq m_j$. Hence $g_P(n_1) + \cdots + g_P(n_\ell) \leq m_{n_1} + \cdots + m_{n_\ell} \leq m_n + \ell - 1 \leq m + \ell - 1$. Thus P is, indeed, (m, n)-good. □

4 Structural Diagonalisation

Theorem 8 from the previous section gives a sufficient condition for the inclusions $\mathrm{V}(m, n) \subseteq \mathrm{V}(h, k)$ and $\mathrm{V}_{\mathrm{fa}}(m, n) \subseteq \mathrm{V}_{\mathrm{fa}}(h, k)$: It suffices that all (m, n)-good k-pools have size at most h. For the recursive case, it is known [7] that this is also a *necessary* condition. This reduces the inclusion and hence also the equality problem for verboseness classes to finite combinatorics.

We now prove that the inclusion structure of fa-verboseness classes is the same as of verboseness classes, by showing that the condition is *also necessary* if we use finite automata instead of Turing machines. That is, we show that if there exists an (m, n)-good k-pool P of size at least $h+1$, then there exists a language in $\mathrm{V}_{\mathrm{fa}}(m, n)$ that is not an element of $\mathrm{V}_{\mathrm{fa}}(h, k)$. Actually our structural diagonalisation will show that the language is not even an element of the much larger class $\mathrm{V}(h, k)$, thus yielding the strong class separation result in Theorem 12. In the following, P will always denote an (m, n)-good k-pool of size $h + 1$.

For finite automata the diagonalisation technique used in [7] does not work, as it involves the usage of universal Turing machines together with finite injury arguments. Instead of these, our *structural diagonalisation* uses a variant of the *k-branches* introduced by Kummer and Stephan [13] for the study of frequency computations. Using only a finite automaton, we can diagonalise along such branches against all Turing machines. We call the diagonalisation *structural* as it uses purely structural properties of the involved languages, and does not depend on computational power at all.

The diagonalisation roughly works as follows: In the tree P^*, see the next paragraph, we construct an infinite branch B and associate a language L_B with it, see Definition 9. Under appropriate assumptions, see Theorem 10, we shall be able to construct a branch B such that $L_B \notin \mathrm{V}(h, k)$. In the proof of $L_B \notin \mathrm{V}(h, k)$ we show that all Turing machines that could possibly witness $L_B \in \mathrm{V}(h, k)$ must *err on some k-tuple of words*. In the construction we associate a k-tuple of words with each node of the tree P^*, and the i-th witness machine will err (at least) on the k words associated with the i-th node on the branch. However, the clever definition of L_B will still allow us to prove $L_B \in \mathrm{V}_{\mathrm{fa}}(m, n)$ in Theorem 11.

The set P^* of words over the 'alphabet' P together with the prefix relation forms an infinite tree. The root of this tree is the empty word, and the successors of a node are obtained by appending different bitstrings from P. An *infinite*

branch $(u_0, u_1, u_2, \dots)$ of P^* is a sequence of nodes $u_i \in P^*$ such that the first node u_0 is the root of P^*, and each u_{i+1} is a successor of u_i. As stated earlier, with each node $u \in P^*$ we wish to associate k different words. We obtain them by adding k different *tags* to u at the front. We use, say, $\mathrm{tag}_i := 0^i 1^{k-i}$ as tags.

Definition 9. *Let P be a k-pool and let $B = (u_0, u_1, u_2, \dots)$ with $u_{i+1} = u_i b_i$ and $b_i \in P$ be an infinite branch of P^*. The language $L_B \subseteq \{0,1\}^*$ contains only words associated with the nodes on this branch, namely those for which*

$$\chi^k_{L_B}\big(\mathrm{tag}_1 u_i, \dots, \mathrm{tag}_k u_i\big) = b_i.$$

Note that just from knowing that a branch B goes through a node u, *we can derive the characteristic values $\chi_{L_B}(w)$ of all words w associated with nodes shorter than u.* A more refined form of this observation will be used in the proof of Theorem 11 to show that all L_B are (m,n)-fa-verbose for appropriate m and n.

Theorem 10. *Let P be a k-pool of size $h+1$. Then there exists a branch B of P^* with $L_B \notin \mathrm{V}(h,k)$.*

Proof. Let $M_0, M_1, M_2, \dots$ be an enumeration of all Turing machines that could possibly witness $L_B \in \mathrm{V}(h,k)$. We construct the branch in stages. Let the node u_i already be chosen. We must decide how to extend the branch to $u_{i+1} = u_i b_i$. Let Q be the set of bit strings enumerated by M_i on input $(\mathrm{tag}_1 u_i, \dots, \mathrm{tag}_k u_i)$. If $|Q| > h$, the machine M_i does not witness $L \in \mathrm{V}(h,k)$ for any language L and we can define $b_i \in P$ arbitrarily. If however $|Q| \le h$, let $b_i \in P \setminus Q$ be any bit string. This concludes the i-th stage. By construction, each machine M_i is now tricked for the words $(\mathrm{tag}_1 u_i, \dots, \mathrm{tag}_k u_i)$ and hence the language L_B induced by the branch $B = (u_0, u_1, u_2, \dots)$ is not an element of $\mathrm{V}(h,k)$. □

Using Theorem 10, we can easily generate a candidate for a set in $\mathrm{V}_{\mathrm{fa}}(m,n) \setminus \mathrm{V}(h,k)$. The theorem supplies us with a language not in $\mathrm{V}(h,k)$. The hard part is to prove that it lies in $\mathrm{V}_{\mathrm{fa}}(m,n)$.

Theorem 11. *Let P be an (m,n)-good k-pool with $0^k \in P$. Then all languages induced by infinite branches of P^* are in $\mathrm{V}_{\mathrm{fa}}(m,n)$.*

Proof. Let L be induced by a branch. We construct an automaton A that on input of any n words $w_1, \dots, w_n$ outputs a pool Q of size at most m containing $\chi^n_L(w_1, \dots, w_n)$.

As a side activity while scanning the input words, A will check which input words are of the form $\mathrm{tag}_t u$ for some t and some node $u \in P^*$. We shall focus on these words in the following, as words that do not pass this syntax check are not elements of L.

As its first action the automaton scans the tags of the words and remembers them in its state. Let $\{u_1, \dots, u_\ell\}$ be the input words with the tags removed. We say that w_j is *associated with the node u_i*, if $w_j = \mathrm{tag}_t u_i$ for some t. Let n_i be the number of words associated with u_i. The next aim of the automaton is to find out how the nodes u_i are related.

The automaton A always reads k bits of all words as a block. For each input word the read bit string is part of the sequence of bit strings that make up the node u_i to which the word is associated. In its state the automaton keeps track of a partition of the node indices $\{1, \ldots, \ell\}$, which serves as a reminder of which u_i have already been identified to lie on different branches of P^*. Initially, all indices are grouped into one large equivalence class. Each time a block has been read, the automaton checks whether there are two indices in the same equivalence class for which the read bit strings now differ. If so, the automaton remembers these indices together with the partition at that moment plus the bit strings themselves. The bitstrings may be thought of as the different 'directions' the branches through the predecessor node headed. Furthermore, the automaton splits the equivalence classes such that only those indices remain equivalent for which it read the same 'direction'.

When the automaton reaches the end of the input words, it will have stored in its state the *forest structure* of the nodes $\{u_1, \ldots, u_l\}$. It will know which nodes are ancestors of which nodes, and which nodes are roots of this forest. If a node u_j is an ancestor of a node u_i, the automaton will also know in which direction a branch through u_i heads when going through u_j.

Consider the (uncountably many) infinite branches B of P^*. Each induces a language L_B in the sense of Definition 9, which in turn induces a characteristic string $\chi^n_{L_B}(w_1, \ldots, w_n)$. This characteristic string depends only on the relative positions of the u_i and on the tags of the input words. Hence, from its knowledge of these relative positions the automaton can produce a pool Q containing *all* induced $\chi^n_{L_B}(w_1, \ldots, w_n)$ and thus also including $\chi^n_L(w_1, \ldots, w_n)$.

It remains to show $|Q| \leq m$. For branches B going through *none* of the nodes u_i, by construction of L_B the characteristic value of $\chi^n_{L_B}(w_1, \ldots, w_n)$ is 0^n. This will be the first bit string found in Q. Consider a root node u_i, that is a node that is not a descendant of another node u_j, and consider all branches B going through u_i, *but going through no other node* u_j. These branches induce *at most* $g_P(n_i)$ *different bit strings in* Q. However, as the branches going through u_i heading off in the 'direction' 0^k all induce the bit string 0^n once more, the branches solely going through u_i induce at most $g_P(n_i) - 1$ *new* bit strings in Q apart from 0^n.

Now consider a direct descendant u_j of a root node u_i and the branches going exactly through u_j and u_i. The branches will induce $g_P(n_j)$ many bit strings in Q, but once more one of them will already be found in Q, namely the one induced by branches that go through u_j and u_i, but which head off in the direction 0^k in u_j. Using a structural induction argument, one can now show that *for each node* u_i *the set of branches going exactly through* u_i *and its ancestors induces at most* $g_P(n_i) - 1$ *new bit strings in* Q. In total we get

$$\begin{aligned} |Q| &\leq 1 + \big(g_P(n_1) - 1\big) + \big(g_P(n_2) - 1\big) + \cdots + \big(g_P(n_\ell) - 1\big) \\ &\leq 1 + m + \ell - 1 - \ell = m, \end{aligned}$$

where we used the (m, n)-goodness of P in the last line. □

Theorem 12. *The following statements are equivalent:*

1. *All (m,n)-good k-pools have size at most h.*
2. $\mathrm{V}(m,n) \subseteq \mathrm{V}(h,k)$.
3. $\mathrm{V}_{\mathrm{fa}}(m,n) \subseteq \mathrm{V}_{\mathrm{fa}}(h,k)$.
4. $\mathrm{V}_{\mathrm{fa}}(m,n) \subseteq \mathrm{V}(h,k)$.

Proof. Statement 1) implies both 2) and 3) by Theorem 8. As we trivially have $\mathrm{V}_{\mathrm{fa}}(m,n) \subseteq \mathrm{V}(m,n)$ for all m and n, both 2) and 3) imply 4). To show that 4) implies 1) we argue by contraposition. If there exists an (m,n)-good k-pool of size $h+1$, there also exists one containing 0^k and then by Theorem 10 there exists a language outside $\mathrm{V}(h,k)$ that is in $\mathrm{V}_{\mathrm{fa}}(m,n)$ by Theorem 11. □

5 Application to Protocol Testing

Beigel's Nonspeedup Theorem [3] states $\mathrm{V}(n,n) = \mathrm{V}(1,1)$, that is all (n,n)-verbose languages are recursive. As this is a statement about inclusion of verboseness classes, we immediately know that the finite automata version $\mathrm{V}_{\mathrm{fa}}(n,n) = \mathrm{V}_{\mathrm{fa}}(1,1)$ also holds, that is *all (n,n)-fa-verbose languages are regular*. We now show how this Finite Automata Nonspeedup Theorem applies to protocol testing.

A simple problem from protocol testing is the following: We are asked to construct a testing device that monitors n signal lines. The device is synchronously fed as input the n symbols currently transported on the different lines, and it should check whether all symbol streams are *valid* with respect to some protocol. When the streams end, the device should tell us on which lines the protocol was not adhered to.

A simple protocol might be 'the stream may not contain the same symbol four times in succession'. If the symbols on the streams represent voltages, this protocol might be used to verify that the signal lines are free of direct currents. A much more complicated protocol is 'the stream consists of valid Internet protocol packets'. It might be used to test an Internet router with numerous signal lines going through it.

If the protocol is sufficiently simple, we can use a finite multi-tape automaton as described in this paper to perform the checking. For example, the simple symbol repetition protocol can obviously be checked by such an automaton. However, for complicated protocols where the set of valid streams is not regular, we cannot hope to use a finite automaton for our testing. This is rather unfortunate, as the high speed used on most signal lines typically forces the use of a simple online device – like a finite automaton.

To overcome the difficulty one might attempt to employ a mixed strategy: We use a finite automaton *plus another simple special purpose hardware that also monitors the signal lines*. For example, such a special purpose hardware might be a counter that is increased every time an 'open' symbol is transported on some line, and decreased every time a 'close' symbol is transported. When the streams end, the special hardware device could communicate some information

to the finite automaton, which should then decide which signal lines were faulty. The special hardware might tell the automaton whether the 'open' and 'close' symbols paired up correctly, or it could tell the automaton the index of a line where this was not the case.

Surely such a special hardware should allow us to check some nonregular protocols. However, the Finite Automata Nonspeedup Theorem tells us *that the special hardware must communicate at least* $\lfloor \log n \rfloor + 1$ *bits to the automaton to be of any use*. To see this, assume that only $\log n$ bits were communicated. Then another automaton, without knowing these $\log n$ bits, could still at least come up with $2^{\lfloor \log n \rfloor} \leq n$ *possibilities* for the set of faulty lines. But then the set of valid streams would be in $\mathrm{V}_{\mathrm{fa}}(n, n)$ and would hence be regular, thus making the special hardware superfluous. In particular, even getting the index of one faulty line does not help the automaton at all.

6 Conclusion

We showed that the inclusion structure of verboseness classes is the same for Turing machines and for finite automata. It is distinct from the corresponding inclusion structure for computational models 'in between', like polynomial time. In particular, the Nonspeedup Theorem holds both for Turing machines and for finite automata, but does not hold for polynomially time-bounded machines. The proofs are based on two new ideas. First, our key lemma states that $f \times g \in \mathrm{EN}(n+m)$ implies $f \in \mathrm{EN}(n)$ or $g \in \mathrm{EN}(m)$; and likewise for finite automata. Second, we introduced the notion of structural diagonalisation, where finite automata trick Turing machines, and used it to show the strong class separations of Theorem 12. While the original Nonspeedup Theorem has limited practical applications, we showed how its finite automata version allows our saving on hardware in finite automata protocol testing.

The parallels between Turing machines and finite automata with respect to partial information algorithms do not appear to end with the inclusion structure of verboseness classes. For example, borrowing the notation from [13], in [1, 2] the classes $\Omega(m, n)$ and $\Omega_{\mathrm{fa}}(m, n)$ are studied. For languages in the first class a Turing machine (and for the second, a finite automaton) can output for any n different words a bit string that agrees on at least m positions with the characteristic string of the words. Although the inclusion problem for $\Omega(m, n)$ is not solved, it turns out that all *known* inclusions of $\Omega(m, n)$-classes also hold for the corresponding $\Omega_{\mathrm{fa}}(m, n)$. Using structural diagonalisation, one can even show strong class separations for these classes similar to the ones established in this paper.

Such similarities lead to a conjecture concerning a powerful extension of the Nonspeedup Theorem, namely Kummer's Cardinality Theorem [12]. This theorem states that $\#_n^A \in \mathrm{EN}(n)$ iff A is recursive, where $\#_n^A(w_1, \ldots, w_n) := \sum_{i=1}^{n} \chi_A(w_i)$. We conjecture that $\#_n^A \in \mathrm{EN}_{\mathrm{fa}}(n)$ iff A is regular. This conjecture can also be rephrased 'in terms of protocol testing': Suppose we are given a finite multi-tape automaton monitoring n input streams, which gets at most $\log n$ bits

of information from an external source. We conjecture that if the automaton can just tell us the *number* of faulty lines, then the set of valid streams must be regular.

Acknowledgments

I am grateful to Lane Hemaspaandra, Arfst Nickelsen, Birgit Schelm, Dirk Siefkes, and Leen Torenvliet for helpful discussions. Furthermore, I would like to thank Leen for suggesting the name 'structural diagonalisation'.

References

1. H. Austinat, V. Diekert, and U. Hertrampf. A structural property of regular frequency classes. *Theoretical Comput. Sci.*, to appear 2002.
2. H. Austinat, V. Diekert, U. Hertrampf, and H. Petersen. Regular frequency computations. In *Proc. RIMS Symposium on Algebraic Systems, Formal Languages and Computation*, volume 1166 of RIMS Kokyuroku, pages 35–42, Research Institute for Mathematical Sciences, Kyoto University, Kyoto, 2000.
3. R. Beigel. *Query-limited reducibilities.* PhD thesis, Stanford University, Stanford, USA, 1987.
4. R. Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Comput. Sci.*, 84(2):199–223, 1991.
5. R. Beigel, W. I. Gasarch, J. Gill, and J. C. Owings. Terse, superterse, and verbose sets. *Inform. Computation*, 103(1):68–85, 1993.
6. R. Beigel, M. Kummer, and F. Stephan. Approximable sets. *Inform. Computation*, 120(2):304–314, 1995.
7. R. Beigel, M. Kummer, and F. Stephan. Quantifying the amount of verboseness. *Inform. Computation*, 118(1):73–90, 1995.
8. J. Cai and L. A. Hemachandra. Enumerative counting is hard. *Inform. Computation*, 82(1):34–44, 1989.
9. W. I. Gasarch and G. A. Martin. *Bounded Queries in Recursion Theory.* Birkhäuser, 1999.
10. C. G. Jockusch, Jr. *Reducibilities in Recursive Function Theory.* PhD thesis, MIT, Cambridge, Massachusetts, 1966.
11. E. B. Kinber. Frequency computations in finite automata. *Cybernetics*, 2:179–187, 1976.
12. M. Kummer. A proof of Beigel's cardinality conjecture. *J. Symbolic Logic*, 57(2): 677–681, 1992.
13. M. Kummer and F. Stephan. Some aspects of frequency computation. Technical Report 21/91, Universität Karlsruhe, Fakultät für Informatik, Germany, 1991.
14. M. Kummer and F. Stephan. Effective search problems. *Math. Logic Quarterly*, 40:224–236, 1994.
15. M. Ogihara. Polynomial-time membership comparable sets. *SIAM J. Comput.*, 24(5):1068–1081, 1995.
16. T. Tantau. Combinatorial representations of partial information classes and their truth-table closures. Diploma thesis, Technische Universität Berlin, Germany, 1999. Available at the Electronic Colloquium on Computational Complexity.