Language Recognition by Nonconstructive Finite Automata

Kaspars Balodis, Rūsiņš Freivalds, Lauma Pretkalniņa, Inga Rumkovska, Madars Virza

Department of Computer Science, University of Latvia, Raiņa bulvāris 29, Riga, LV-1459, Latvia

Abstract. Nonconstructive finite automata were first considered by R. Freivalds in [1]. We prove tight upper bound for amount of nonconstructivity that can be needed to recognize a language. We prove some theorems about saving amount of the nonconstructive help needed by encoding that information in automata. We also show that nonconstructive probabilistic automata can be more concise than nonconstructive deterministic automata.

1 Introduction

Nonconstructive proofs shows the existence of mathematical object with certain properties not by showing how to construct it but merely by proving it is impossible that the object could be nonexistent. The history of nonconstructive proofs as well as the related controversy goes back to the end of the nineteenth century and early twentieth century. Paul Gordan was known for producing highly complicated constructive proofs in invariant theory. Still after all of his achievements there were some important problems left open. David Hilbert seems to be one of the first who used nonconstructive proofs in mathematics. He decided to take a completely different approach to manage the problems considered by Gordan and used proof by contradiction to show that there is no way solutions fail to exist. For Gordan it was hard to accept that a proof without constructions could be called mathematics. He objected publishing Hilbert's papers and even called them "theology", however despite his objections these papers were published in [6].

Later, in forties, nonconstructive proofs finally found their way in mathematics. Paul Erdös even used them in discrete mathematics [9].

R. Freivalds introduces a method of how to measure the amount of used nonconstructivity in [1]. We will base our research on this method. We will use the definition of automaton nonconstructively recognizing the language from [1].

Definition 1. *We say that an automaton A recognizes the language L nonconstructively if the automaton A has an input tape where a word x is read and an additional input tape for nonconstructive help y with the following property. For arbitrary natural number n there is a word y such that for all words x whose*

length does not exceed n the automaton A on the pair (x, y) produces the result 1 if $x \in L$, and A produces the result 0 if $x \notin L$. Technically, the word y can be a tuple of several words and may be placed on separate additional input tapes.

We will use the definition of nonconstructivity measure from [1] as well.

Definition 2. *We say that an automaton A recognizes the language L nonconstructively with nonconstructivity $d(n)$ if the automaton A has an input tape where a word x is read and an additional input tape for nonconstructive help y with the following property. For arbitrary natural number n there is a word y of the length not exceeding $d(n)$ such that for all words x whose length does not exceed n the automaton A on the pair $(x.y)$ produces the result 1 if $x \in L$, and A produces the result 0 if $x \notin L$. Technically, the word y can be a tuple of several words and may be placed on separate additional input tapes. In this case, $d(n)$ is the upper bound for the total of the lengths of these words.*

Notion similar to amount of nonconstructivity has already been studied - R. Karp and R. Lipton in [7] have introduced a notion *Turing machine that takes advice* which is practically the same notion for Turing machines as ours *nonconstructive computation by finite automata*. Later C. Damm and M. Holzer have adapted the notion of advice for finite automata. Since there was no reference to intuitionism in [7], the adaptation was performed in the most straightforward way (what is quite natural). However the notion of *finite automata that take advice* in [5] differs from our notion very much. These notions are equivalent for large amounts of nonconstructivity (or large amounts of advice) but, for the notion introduced in [5] and later extensively used by T.Yamakami and his coauthors [12, 8, 11], languages recognizable with polynomial advice are the same languages which are recognizable with a constant advice. Our notion of the amount of nonconstructivity is such that our most interesting results concern the smallest possible amounts of nonconstructivity.

A similar situation was in sixties of the 20th century with space complexity of Turing machines. At first space complexity was considered for one-tape offline Turing machines and it turned out that space complexity is never less than linear. However, it is difficult to prove such lower bounds. Then the seminal paper by R.E.Stearns, J.Hartmanis and P.M.Lewis [10] was published and many-tape Turing machines became a standard tool to study sublinear space complexity.

Some bounds of the above mentioned nonconstructivity measure are proved in the [1].

Theorem 1. *If a language L can be nonconstructively recognized with a nonconstructivity bounded by a function $d(n) = o(\log n)$, then L is regular.*

Theorem 2. *There exists a nonrecursive language L and a function $g(n)$ such that L can be nonconstructively recognized by a DFA with a nonconstructivity $g(n) \in polylog(n)$.*

Theorem 3. *There exists a nonregular language L and a function $g(n)$ such that L can be nonconstructively recognized with nonconstructivity $g(n)$ and $\log n \leq g(n) \leq (\log n)^2$.*

These three theorems show what is the smallest necessary amount of nonconstructivity of recognizing any nonregular or nonrecursive language. In our paper we are stating the bounds for nonregular languages more precisely. We also prove tight upper bound for amount of nonconstructivity that can be needed to recognize any language. We also investigate language recognition with multi-tape finite automata.

In R. Freivalds' paper [1] he mainly considers two-way deterministic finite automata with nonconstructivity. Two-way deterministic finite automata (with nonconstructivity) will be our considered computational model for most cases as well.

2 Results on finite automata

Theorem 4. *If a language L can be nonconstructively recognized with nonconstructivity $f(n)$ then for every constant C this language can be nonconstructively recognized with nonconstructivity $g(n)$ such that for infinitely many n: $g(n) \leq f(n) - C$ (technically $g(n) \leq max(f(n) - C, 0)$).*

Proof. If a language L can be nonconstructively recognized with nonconstructivity $f(n)$ it means that there exists a help word list $y_0, y_1, y_2, \ldots$ such that $|y_n| \leq f(n)$ and y_n can be used as a help word for every recognizable word x whose length does not exceed n. If for every word in this list the first C symbols would be the same, we could encode them into the automata and use a shorter help word.

Unfortunately, it might be that the help word list doesn't have this property. However, if there are infinitely many help words not shorter than C, then there exists an index list $i_0, i_1, i_2, \ldots$ such that all help words from the list $y_{i_0}, y_{i_1}, y_{i_2}, \ldots$ begin with the same C symbols as there are only finite number of different beginnings of length C (if there are not infinitely many help words not shorter than C, then the nonconstructivity used by automata is bounded by a constant therefore it can be entirely encoded in the automata). We can define a new help word list for the same automata $y'_k = y_{i_j}$, where j is the minimal for which $i_j \geq k$. This new help word list $y'_0, y'_1, y'_2, \ldots$ not only works as well as the old help word list (from definition that help word y_n must work for all words x ($|x| \leq n$)), but also has the property that the first C symbols of all help words are the same. For infinitely many n: $|y'_n| = |y_n|$. If we encode the first C symbols into automata and remove them from help words, thus obtaining new help word list $y''_0, y''_1, y''_2, \ldots$, then for infinitely many n: $|y''_n| \leq |y_n| - C$ □

Theorem 5. *There exists a language L in a binary alphabet such that it cannot be nonconstructively recognized by a DFA with a nonconstructivity less than $\Omega(2^n)$.*

Proof. P.Martin-Löf in [3] proved that there exists an infinite sequence $a_1, a_2, a_3, \cdots$ such that infinitely many initial fragments of it have Kolmogorov complexity n and all the initial fragments of it have Kolmogorov complexity no less than

$n - O(\log_2 n)$ (he also proved that there are no infinite binary sequences with a higher Kolmogorov complexity). If a string of length n has Kolmogorov complexity of $O(f(n))$, that means that there is no way of describing that string by using less than $O(f(n))$ bits of information.

Let $ch(w)$ be the characteristic function of language L:

$$ch(w) = \begin{cases} 0, \text{ if } w \in L \\ 1, \text{ if } w \notin L \end{cases}$$

We define this language as:

$$ch(w_i) = i\text{-th symbol in P.Martin-Löf sequence}$$

Where w_i is the i-th binary word in ordering $\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$.

Assume from the contrary that there exists a nonconstructive deterministic finite automaton such that for all values of n the nonconstructivity is less than $\Omega(2^n)$. From the given program of the automaton and from the given nondeterministic help y_n one can algorithmically reconstruct the values $a_1, a_2, a_3, \ldots, a_{2^n}$ of P.Martin-Löf sequence. Hence Kolmogorov complexity of the initial fragment $a_1, a_2, a_3, \ldots, a_{2^n}$ for every n exceeds the length of the nonconstructive help y_n no more than by a constant and this initial fragment of length 2^n has Kolmogorov complexity less than $\Omega(2^n)$. But this is contradiction to properties of P.Martin-Löf sequence. □

Theorem 6. *Every language L over the binary alphabet $0, 1$ can be recognized nonconstructively with a nonconstructivity $O(n2^n)$.*

Proof. The idea is to encode the language as word/answer pairs in the help word.

We will use the ternary alphabet $0, 1, 2$ for our help word. Suppose that our help word will have to work for all inputs words of length not exceeding n. The help word will be the concatenation of "$w2a_w2$" for all words w of length not exceeding n, where a_w is defined to be 1 if w is in L and 0 otherwise. The order in which these strings are to be concatenated does not matter to our automaton.

The automaton will work by comparing the input word with all words encoded in the help word and returning the answer as soon as it can be determined.

Each of the possible input words on the help word tape has length equal or less than to n. There are exactly $2^n + 2^{n-1} + \ldots + 1 = 2^{n+1} - 1$ possible input words in the help word together with $2^{n+1} - 2$ boundary markers. Therefore the total length of the help word is $O(n2^n)$.

The bound given above is provided for its simplicity, however it doesn't close the gap with lower bound in theorem 5.

The following theorem uses a result by Camille Flye Sainte-Marie, which was further generalized by Nicolaas Govert de Bruijn and Tanja van Ardenne-Ehrenfest.

Theorem 7 (De Bruijn sequence). *For every positive integer n and every alphabet A with size k there exists a sequence $B(n, k)$ with length $n^k + k - 1$, that contains every k character subsequence over A exactly once. [13]* □

For example, the sequence "0001011100" is de Bruijn sequence $B(2,3)$, because it contains all three digit sequences 000, 001, ..., 110, 111 as subsequences.

Theorem 8. *Every language L over the binary alphabet can be recognized nonconstructively with a nonconstructivity $O(2^n)$.*

Proof. The idea is to do a space-efficient encoding of all possible input words.

The help word will be taken over the ternary alphabet.

For a fixed de Bruijn sequence $B(k,2)$ we define $H(k)$ to be a binary sequence of length $2(2^k+k-1)$ with its i-th term defined to be:

$$H(k)_{2i-1} = B(k,2)_i$$

$$H(k)_{2i} = \begin{cases} 0, \text{ if } i < k \text{ or } B(k,2)_{i-k+1} \cdots B(k,2)_i \text{ is not in } L \\ 1, \text{ otherwise} \end{cases}$$

This definition means that $H(k)$ essentially has every k digit sequence S as a subsequence of its odd-numbered terms, moreover, having located S we can easily check if S is in L, by the definition of $H(k)$'s even-valued terms.

We define our help word as $H(0)2H(1)2\cdots 2H(n)$.

For input word of length i automaton can position itself on the first character of $H(i)$ by skipping over i 2's. After this step automaton continues by checking the odd-numbered characters of $H(i)$ to find the input word. When the input word is found as a subsequence of $H(i)_j$ terms for odd j, the automaton reads the answer, which is exactly the next character on help word. □

For example, the meaning of $H(3)$ for $B(3,2) = 0001011100$ is as follows:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
H(3)_i	**0**	0	**0**	0	**0**	a	**1**	b	**0**	c	**1**	d	**1**	e	**1**	f	**0**	g	**0**	h

always zero for H(3)

a is 1 if "000" is in L, 0 otherwise
b is 1 if "001" is in L, 0 otherwise
c is 1 if "010" is in L, 0 otherwise
d is 1 if "101" is in L, 0 otherwise
....
i is 1 if "100" is in L, 0 otherwise

This construction can be extended for languages over arbitrary alphabets, which leads us to the following result:

Theorem 9. *Every language L over the alphabet of size k can be recognized nonconstructively with a nonconstructivity $O(k^n)$.*

3 Results on multi-tape finite automata

Theorem 10. *There exists a nonregular language in binary alphabet that can be recognized nonconstructively using nonconstructivity $O(\ln n)$ on two tapes.*

Proof. We consider language

$$L = \{0^{p_1}10^{p_2}10^{p_3}1\ldots10^{p_m}110^k \mid \exists n \geq 1\ k = n \cdot p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_m\}$$

where p_i is the i-th prime. If we used one help-word

$$0^{p_1}10^{p_2}10^{p_3}1\ldots10^{p_n}$$

with $n \geq m$ then automata could check if the recognizable word has the needed prefix $0^{p_1}10^{p_2}10^{p_3}1\ldots10^{p_m}$. Also automata could check if the k in the 0^k part is divisible by all p_i $(1 \leq i \leq m)$ by positioning the help-word's head on the beggining of m-th prime and then checking if the 0^k divides by p_m. If this test succeeds, then proceed to the previous prime (in this case p_{m-1}) and so continue until we have checked the divisibility by all primes $p_m, p_{m-1}, \ldots, p_1$.

The product of first m primes is called *primorial of* m and it is known that $Primorial(m) \approx e^{m \ln m}$ (see e.g. [4]). The length of the help-word that helps for all words starting with the prefix $0^{p_1}10^{p_2}10^{p_3}1\ldots10^{p_m}11$ is

$$\Sigma(m) = \Sigma_{i=1}^{m} p_i.$$

It is known that

$$\Sigma(n) \approx \frac{1}{2}n^2 \ln n.$$

So the length of help-word is between $\ln n$ and $\ln^2 n$, where n is the length of recognizable word.

We can improve this length to $\ln n$ by using two nonconstructive tapes. On one tape there will be sequence of length p_n defined in this way:

$$S(i) = \begin{cases} 1, \text{ if } i \text{ is prime} \\ 0, \text{ otherwise} \end{cases}$$

and on the other tape there will be array of ones with length n. As it is known that $p_n \approx n \cdot \ln n$, the overall length of nonconstructive tapes is $p_n + n \approx n \cdot \ln n = O(n \ln n)$.

So, for the recognizable word of length $\approx e^{m \ln m}$ we need $O(m \ln m)$ nonconstructivity. This corresponds to recognizable word length n and nonconstructivity $O(\ln m)$.

The automata with two nonconstructive tapes can check if the length of 0^k is divisible by p_m by positioning it's recognizable word's head on beggining of 0^k, the first nonconstructive tape's head on p_m and second tape's head on the first symbol of the tape. Then simultaneously moving forward recognizable word's head and moving backward first nonconstructive tape's head, and everytime when on the first nonconstructive tape is symbol "1" then move forward head

of second nonconstructive tape. So always keeping the count of "1"'s to the left of both nonconstructive heads the same (because everytime we move past "1" with the first head, we move forward the second head).

When we have got to the beginning of the first nonconstructive tape, we will be in position m on the second tape. To get to the starting position (position on first tape: p_m, on second: 0), we can move forward the first head and everytime we encounter an "1" on the first tape, we move backward the head on the second tape, till we are at p_m on first tape and at 0 on the second tape.

In such way, we can check the divisibility by p_m and at the end of check be at the starting position (first tape: p_m, second tape: 0). Then we can position the head on previous prime and check if the recognizable 0^k is divisible by it and so continue till all primes are considered. □

Theorem 11. *Any language which can be recognized with a Turing Machine with space complexity $f(n)$ can be recognized nonconstructively with DFA with nonconstructivity $const^{f(n)}$ on 4 tapes.*

Proof. Turing Machine can be simulated by DFA with 2 stacks. See for example [14] The whole content of work-tape is contained into two stacks - everything on the left side of Turing Machine's head in the first one, and everything on the right side - in the second one. So the content of the stack doesn't exceed the space used by Turing Machine.

Stack can be simulated with 2 counters. [14] We can assume, without loss of generality, that stack alphabet is binary, because we can encode any symbol from longer alphabet as sequence of several binary symbols. The content of the stack can be regarded as a natural number's binary representation with the least significant bit at the top of stack. We can store this number into our first counter. Then pushing 0 in the stack is the same as multiplying this number by 2. This can be done by decreasing the first counter by 1 and adding 2 to the second counter until the first counter is 0. Then we can transfer this number from the second counter to the first one. Pushing 1 in the stack is the same with the difference that at the end the counter's value is incremented by one. Popping out a digit from stack is the same as dividing the number by two and calculating remainder. This can be done by decreasing the first counter by 2 and adding 1 to the second counter until the first counter is 0 and then transferring second counter's value into the first counter. It is easy to check whether 0 or 1 was popped out.

If we can simulate Turing Machine with 2 stacks and stack with 2 counters, then we can simulate Turing Machine with 4 counters.

The maximum number in the counter is $const^{f(n)}$ where const depends on size of alphabet and $f(n)$ is the space used by Turing Machine.

It is easy to see that a counter can be simulated with an empty tape with length at least maximum number that can be stored in counter.

Therefore we can simulate Turing Machine with space complexity $f(n)$ with nonconstructive automata with nonconstructivity $const^{f(n)}$. □

4 Results on probabilistic finite automata

Similarly as two-way DFA with nonconstructivity we can also consider two-way probabilistic automata with nonconstructivity.

Theorem 12. *There exists a nonregular language L that can be nonconstructively recognized with probabilistic automata with constant nonconstructivity. DFA with nonconstructivity recognizes the language L with nonconstructivity $g(n)$, where $\log n \leq g(n) \leq \log^2 n$. Thus probabilistic nonconstructive automata can use smaller amount of nonconstructivity than deterministic automata.*

Proof. Let's consider such $L = \{0^k 1^k\}$.

Freivalds in paper [2] proved that L can be recognized with two-way probabilistic automaton. Thus, if we look at such automaton as automaton with nonconstructivity, it takes constant (zero sized) amount of nonconstructivity. As this language is not regular, a nonconstructive DFA must take bigger amount of nonconstructivity than $o(\log n)$ (it follows from theorem 1) [1].

Language L can be recognized with nonconstructivity $g(n)$, where $\log n \leq g(n) \leq (\log n)^2$. We will use $0^{p_1}10^{p_2}10^{p_3}1\ldots10^{p_m}$ as help-word, where $p_1, p_2, p_3, \ldots, p_m$ is first m primes. Automata compares length of arrays of 0 and 1 by comparing their lengths modulo p_i for each p_i given in help-world. If the lengths are equal for all modules $p_1, p_2, p_3, \ldots, p_m$ then the lengths are equal or difference between them is at least $p_1 \cdot p_2 \cdot \ldots \cdot p_m$. Thus the given help-word works correctly if size of input word $n \leq p_1 \cdot p_2 \cdot \ldots \cdot p_m$. This product is called *primorial of* m and it is known that $Primorial(m) \approx e^{m \ln m}$. The length of the help-word is $\sum_{s=1}^{m} p_s \approx \frac{1}{2} n^2 \ln n$. Thus, if $n = e^{m \ln m}$, then $\ln n = m \ln m$ and $\ln^2 n = m^2 \ln^2 m$. So $m \ln m < \sum_{s=1}^{m} p_s < \ln^2 n = m^2 \ln^2 m$ and the amount on nonconstructivity in this case is $\log n$ and $\log^2 n$. □

If we consider the probabilistic automata from upper proof as automata without nonconstructivity, then this proof is also interesting because it provides an example how to measure an amount of nonconstructivity of a practical problem. This proof provides the information about amount of nonconstructivity (as measure we will use nonconstructive DFA) of the probabilistic automata recognizing language L. This amount is bigger than $o(\log n)$ and less than $\log^2 n$.

References

1. Rūsiņš Freivalds. Amount of nonconstructivity in finite automata. *Lecture Notes in Computer Science,* vol. 5642, pp. not known yet, 2009.
2. Rūsiņš Freivalds. Probabilistic two-way machines. *Lecture Notes in Computer Science,* vol. 188, pp. 33–45, 1981
3. Per Martin-Löf. The definition of random sequences. *Information and Control,* vol. 9, No. 6, pp. 602-619, 1966.
4. Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory.*, vol. 1, MIT Press, 1996.

5. Carsten Damm and Markus Holzer. Automata that take advice. *Lecture Notes in Computer Science,* vol. 969, pp. 565–613, 1995.
6. David Hilbert. Über die Theorie der algebraischen Formen. *Mathematische Annalen,* Bd. 36, pp. 473–534, 1890.
7. Richard M. Karp and Richard Lipton. Turing machines that take advice. *L' Enseignement Mathematique*, vol. 28, pp. 191–209, 1982
8. Harumichi Nishimura, Tomoyuki Yamakami. Polynomial time quantum computation with advice. *Information Processing Letters*, vol. 90, No. 4, pp. 195–204, 2004.
9. Paul Erdös. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society,* Volume 53, Number 4 (1947), pp. 292–294, 1947.
10. Richard Edwin Stearns, Juris Hartmanis, Philip M. Lewis II. Hierarchies of memory limited computations. *Proceedings of FOCS*, pp.179–190, 1965.
11. Kohtaro Tadaki, Tomoyuki Yamakami, Jack C. H. Lin. Theory of One Tape Linear Time Turing Machines. *Lecture Notes in Computer Science,* vol. 2932, pp. 335–348, 2004.
12. Tomoyuki Yamakami. Swapping lemmas for regular and context-free languages with advice. *The Computing Research Repository (CoRR)*, CoRR abs/0808.4122, 2008.
13. Tanja van Aardenne-Ehrenfest, Nicolaas Govert de Bruijn Circuits and trees in oriented linear graphs *The Bulletin of the Belgian Mathematical Society – Simon Stevin*, vol. 28, pp. 203–217, 1951.
14. Mikhail J. Atallah *Algorithms and theory of computation handbook*, CRC Press, 1998