Electronic Colloquium on Computational Complexity, Report No. 27 (2000)

A Separation of Determinism, Las Vegas and Nondeterminism for Picture Recognition*

Pavol Ďuriš † Juraj Hromkovič † Katsushi Inoue ‡

Abstract

The investigation of the computational power of randomized computations is one of the central tasks of current complexity and algorithm theory. In this paper for the first time a "strong" separation between the power of determinism, Las Vegas randomization, and nondeterminism for a computing model is proved. The computing model considered here are finite automata with two-dimensional input tapes (i.e., finite automata recognizing picture languages). Moreover, a hierarchy on the degree of nondeterminism of two-dimensional finite automata is proved.

1 Introduction

The comparative study of the computational power of deterministic, randomized and nondeterministic computations is one of the central tasks of complexity and algorithm theory. This is not surprising because the current view on the specification of the class of tractable (practically solvable) problems is that a computing problem is tractable if there exists a randomized polynomial-time algorithm for it. In this paper we focus on the power of Las Vegas algorithms that are very practical because they never err. It is not surprising that the central questions like P versus ZPP (polynomial-time Las Vegas computations) and ZPP versus NP are unresolved, because $P \subseteq ZPP \subseteq NP$.

The recent research has focussed on the comparison of Las Vegas with determinism and nondeterminism for restricted models in order to better understand the nature and the power of Las Vegas randomization. We have only few results comparing Las Vegas and determinism, and in most cases determinism and Las Vegas are polynomially related (combinational complexity of Boolean circuit, communication complexity [MS82], the size of finite automata [DHRS97], and the size of OBDDs [DHRS97]). For the

*The work on this paper has been supported by DFG-Project HR 14/3-2.

†Dept. of Computer Science I (Algorithms and Complexity) University of Technology, Aachen Ahornstraße 55, 52056 Aachen, Germany, E-Mail: jh@i1.informatik.rwth-aachen.de

‡Department of Computer Science and Systems Engineering, Faculty of Engineering, Yamaguchi University, Ube, 755-8611 Japan, E-mail: inoue@esse.yamaguchi-u.ac.jp.

ISSN 1433-8092

time complexity of CREW PRAMs [DKR94] and one-way communication complexity [DHRS97] there is even a linear relation between Las Vegas and determinism. On the other hand (may be a little bit surprising) there is an exponential gap between Las Vegas and determinism for the size of one-time-only branching programs [Sa99] and there is an at least superpolynomial gap for the size of k-OBDDs [HSa99]. An exponential gap between nondeterminism and Las Vegas seems to be typical for all known results (one-way finite automata [DHRS97], two-way and one-way communication complexity [AHY83, Hr97, KN97, PS84, Ya79, DHRS97], and OBDDs [Sa97]). One conjectures that there is an exponential gap between Las Vegas and nondeterminism for the size of two-way finite automata too, but up till now only a quadratic gap was proved [HS99].

Here, we for the first time prove a strong separation between nondeterminism, Las Vegas, and determinism, for a computing model. A "strong separation" means that Las Vegas can do something what the determinism cannot do and that nondeterminism can do something what Las Vegas cannot do. Obviously, such a separation can be proved for restricted uniform models only, because for non-uniform models to compute a solution of a given problem is a question of complexity, but not the question of computability. The computing model considered here are finite automata with a two-dimensional squared input, i.e. automata recognizing some picture languages. In contrast to standard finite automata whose nondeterministic, Las Vegas, and deterministic versions recognize the same class of regular languages (and so one can compare the sizes of them only), our results shows that this is not true for finite automata with two-dimensional inputs.

More precisely, we prove that there is a language recognized by two-dimensional Las Vegas finite automata, but not by deterministic ones, for the general model of two-dimensional finite automata (called also four-way model) as well as for the three-way model, where the movement upwards on the two-dimensional input tape is forbidden. The separation between nondeterminism and Las Vegas is proved for the three-way model only. In fact we even prove a stronger result, that there is a language accepted by a three-way two-dimensional nondeterministic finite automaton, but by no three-way two-dimensional $o(n/\log_2 n)$ space-bounded Las Vegas Turing machine.

The second research problem considered in this paper is the investigation of the computational power of nondeterministic computing devices with restricted nondeterminism. There are only few results [DT90, HS96, GKW90, KW80] measuring the computational power of restricted nondeterminism. There are two possibilities to measure the amount of nondeterminism in computation. One possibility is to count the number of advice bits (nondeterministic guesses) in particular nondeterministic computations, and the second possibility is to count the number of different nondeterministic computations on any input. Here, we consider the later one. For three-way two-dimensional finite automata we prove a strong hierarchy - one additional nondeterministic computation can help to recognize a language that cannot be recognized without this small increase of the degree of nondeterminism. This contrasts to one-way finite automata, where one additional nondeterministic guess can be always simulated with a small (polynomial) increase of the number of states.[1] For the general two-dimensional finite automata we

[1] Remember that deterministic finite automata and nondeterministic finite automata recognize the same class of regular languages, and so the simulation cost of an increase of nondeterminism is measured

are not able to prove such a strong hierarchy. We only show that n nondeterministic computations cannot be simulated by $o(n/\log n)$ computations on inputs of size $n \times n$.

This paper is organized as follows. In Section 2 the basic definitions and notations are given. In Section 3 the results are presented and discussed. Because the proof methods essentially change from theorem to theorem, we are not able to present all technical proofs in this extended abstract. To illustrate the proof techniques we give two short sketches of proofs directly in Section 3. Some detailed proof of the separation between Las Vegas and determinism and the proofs of the hierarchies on the degree of nondeterminism are moved to Appendix.

2 Definitions

In what follows we consider two-dimensional finite automata as defined in [IITT82, HIT89]. Informally, a two-dimensional deterministic finite automaton, **2-dfa**, consists of a finite control, a read-only head, and a squared two-dimensional input tape bounded by a special boundary symbol ¢. The set of states of a 2-dfa is divided into three disjoint sets of working, accepting and rejecting states. No action is possible from any rejecting or accepting state. The read-only head can move in all four directions (to the left, to the right, upwards, downwards) from any square, but boundary square containing the special symbol ¢, of the two-dimensional input tape. If one forbids to move upwards, then we speak about three-way 2-dfa, **tr2-dfa**, that are an analogue to one-way finite automata in the one-dimensional case. Let, for every kind of device A, $\boldsymbol{L(A)}$ denote the language accepted by A. The nondeterministic versions of two-dimensional automata are denoted **2-nfa** and **tr2-nfa**.

To define two-dimensional Las Vegas automata, we use the same way as to define the Las Vegas mode for ordinary finite automata [DHRS97, HS99]. A two-dimensional Las Vegas finite automaton A, **2-lvfa**, can be viewed as a 2-nfa with a probability distribution for every nondeterministic branch, and four classes of states: working, accepting, rejecting, a neutral ("I do not know") states. There is no possible move from accepting, rejecting and neutral states. A is not allowed to make mistakes: If there is a computation of A on an (two-dimensional) input x finishing in an accepting [rejecting] state, then x must be in $L(A)$ [x may not be in $L(A)$]. One requires, for every $y \in L(A)$ [$y \notin L(A)$], that A reaches an accepting [rejecting] state with probability of at least $1/2$ (i.e., the probability to reach a neutral state is at most $1/2$ for every input).

We denote by **tr2-lvfa** the three-way version of 2-lvfa. In what follows 2-DFA [TR2-DFA, 2-NFA, TR2-NFA, 2-LVFA, TR2-LVFA resp.] is the family of two-dimensional squared languages accepted by 2-dfa's [tr2-dfa's, 2-nfa's, tr2-nfa's, 2-lvfa's, and tr2-lvfa's respectively].

To be able to conveniently work with the inputs, we give the following definition.

Definition 2.1 Let Σ be a finite set of symbols. A **two-dimensional tape** over Σ is a

in the number of states only.

two-dimensional array of elements of Σ. The set of all the two-dimensional tapes over Σ is denoted by $\Sigma^{(2)}$. Given a tape $x \in \Sigma^{(2)}$, we let $l_1(x)$ be the number of rows and $l_2(x)$ be the number of columns. For each $m, n \geq 1$, let $\Sigma^{m \times n} = \{x \in \Sigma^{(2)} | l_1(x) = m \wedge l_2(x) = n\}$. If $1 \leq i_k \leq l_k(x)$ for $k = 1, 2$, we let $x(i_1, i_2)$ denote the symbol in x with coordinates (i_1, i_2). Furthermore, we define $x[(i_1, i_2), (i'_1, i'_2)]$, only when $1 \leq i_1 \leq i'_1 \leq l_1(x)$ and $1 \leq i_2 \leq i'_2 \leq l_2(x)$, as the two-dimensional tape z satisfying the following (i) and (ii):

(i) $l_1(z) = i'_1 - i_1 + 1$ and $l_2(z) = i'_2 - i_2 + 1$;

(ii) for each $i, j (1 \leq i \leq l_1(z), 1 \leq j \leq l_2(z))$ $z(i, j) = x(i_1 + i - 1, i_2 + j - 1)$.

Definition 2.1 □

3 Results

3.1 Las Vegas versus Determinism

Here we separate determinism and Las Vegas for the general 2-fa as well as for the three-way 2-fa. Note, that this is the first result showing that a Las Vegas computation device can do something that cannot be done by the corresponding deterministic device.

Let $Mid = \{x \in \{0,1\}^{(2)} \,|\, l_1(x) = l_2(x) = 2n+1 \text{ for some } n \in I\!N, \text{ and } x(n+1, n+1) = 1\}$[2]

Theorem 3.1 $Mid \in$ 2-LVFA $-$ 2-DFA. Theorem 3.1 □

Idea of the proof. To prove $Mid \in$ 2-LVFA one has to use a version of the known technique of probabilistic simulation of nondeterministic decisions [Gi77, MS97, HS99]. The fact that $Mid \notin$ 2-DFA was proved in [IITT82].

Theorem 3.1 ■

To make the separation for the three-way case, consider the language

$$\begin{aligned} L = \{ \; & y \in \{0,1,2\}^{(2)} \,|\, \exists n, j, l : l_1(y) = l_2(y) = n \geq 2, 1 \leq j \leq n-1, \\ & 0 \leq l \leq n-3, \text{ the row } j \text{ of } y \text{ is } 10^l 20^{n-l-2}, \\ & \text{every row above the } j\text{-th row is } 10^{n-2}1, \text{ and} \\ & [(n-j-1 \leq l \wedge y(n,n) = 1) \vee (n-j-1 \leq n-l-3 \wedge y(n,n) = 0)]\}. \end{aligned}$$

Note that the inequality $n - j - 1 \leq l$ [the inequality $n - j - 1 \leq n - l - 2$] means that the symbol 2 in the j-th row of y is on or below the secondary [the main] diagonal of y.

[2] i.e., the symbol 1 is in the middle of the picture.

Theorem 3.2 $L \in$ TR2-LVFA $-$ TR2-DFA. Theorem 3.2 □

The full proof of Theorem 3.2 is given in Appendix A.

3.2 Las Vegas versus Nondeterminism

We are able to separate nondeterminism and Las Vegas in the three-way case only. Whether 2-LVFA $\subset$ 2-NFA remains open.

Let

$$\textit{Non-Eq} = \{x \in \{0,1\}^{(2)} \mid \text{ the first row and the second row of } x \text{ are different}\}$$

Theorem 3.3 *Non-Eq* $\in$ TR2-NFA $-$ TR2-LVFA. Theorem 3.3 □

Sketch of the proof. The fact *Non-Eq* $\in$ TR2-NFA is obvious. Running in the first row from the left to the right, a tr2-nfa can nondeterministically guess the position i such that $x(1,i) \neq x(2,i)$ and check this guess by moving downwards from the position $(1,i)$ to the position $(2,i)$.

To proof *Non-Eq* $\notin$ TR2-LVFA one can apply the ideas of communication complexity theory. Cut the picture x horizontally between the first row and the second row. Obviously, any tr2-lvfa working on x crosses at most once this cut because it cannot move upwards. This crossing can be viewed as a communication message (information transfer) from the upper part of the input to the lower part of the input. This communication message contains a description of the current state and the crossing position i, $1 \leq i \leq n$. So, for inputs of size $n \times n$, the communication complexity measured as the binary length of the message is in $O(\log_2 n)$. So, if *Non-Eq* $\in$ TR-LVFA, then there is a Las Vegas one-way communication protocol recognizing *Non-Eq* within the communication complexity $O(\log_2 n)$. But this is impossible, because every deterministic protocol recognizing the non-equality of two binary strings of length n requires the communication complexity n (see, for instance, [Hr97, KN97]) and every one-way Las Vegas protocol has its communication complexity at least one half of the communication complexity of the best deterministic protocol for this task [DHRS97].

Theorem 3.3 ■

3.3 A Hierarchy on the Amount of Nondeterminism

These are two distinct, natural possibilities how to measure the amount of nondeterminism of nondeterministic computations. The first possibility is to consider the number of different computations on a fixed input[3] and the second one is to count the number of guesses in particular computations. Here we consider the first possibility

[3] i.e., to count the number of leaves of nondeterministic computation trees.

and two-dimensional Turing machines (**2-tm**). A 2-tm is a natural extention of a 2-fa by one linear working tape.[4] We use the notations **2-ntm** for a two-dimensional nondeterministic Turing machine, and **tr2-ntm** for the three-way version of 2-ntm.

Definition 3.1 Let $f(n) : I\!N \to I\!N$ be a function. A 2-ntm (tr2-ntm) M is $\boldsymbol{f(n)}$ **space bounded** if for any $n \geq 1$ and for any square input tape x of side-length n, any computation path of M on x does not use more than $f(n)$ cells of the storage tape.
Definition 3.1 □

Definition 3.2 Let $g(n) : I\!N \to I\!N$ be a function. A 2-ntm (tr2-ntm, 2-nfa, tr2-nfa) M is $\boldsymbol{g(n)}$ **path-bounded** if for any $n \geq 1$ and for any square input tape x of side-length n, there are at most $g(n)$ computation paths of M on n. Definition 3.2 □

Notation:

- 2-NTM$(f(n), g(n))$ ... the class of sets (of square tapes) accepted by $f(n)$ space-bounded and $g(n)$ path-bounded 2-ntm's.
- TR2-NTM$(f(n), g(n))$ ... the class of sets (of square tapes) accepted by $f(n)$ space-bounded and $g(n)$ path-bounded tr2-ntm's.
- 2-NFA$(g(n))$ ... the class of sets (of square tapes) accepted by $g(n)$ path-bounded 2-nfa's.
- TR2-NFA$(g(n))$ ... the class of sets (of square tapes) accepted by $g(n)$ path-bounded tr2-nfa's.

First, we consider the three-way case. The following theorem shows that some restrictions on the number of paths (leaves) of nondeterministic computation trees cannot be compensated by an increase of the space complexity.

Theorem 3.4 For any function $f(n) = o(\log n)$, and any function
$g(n) \in \{(\log\log n)^k \mid k \text{ is a positive integer}\} \cup \{r^{\log\log n} \mid r \text{ is a positive integer}\}$,

$$\text{TR2-NTM}(\log\log n, g(n)) - \text{TR2-NTM}(f(n), o(g(n))) \neq \varnothing.$$

Theorem 3.4 □

For finite automata one can obtain the following strong hierarchy. This strongly contrasts to one-way finite automata, where any constant increase of the number of nondeterministic computations can be compensated by a polynomial increase of the number of states.

[4] for formal definitions see, for instance, [IITT82, HIT89].

Theorem 3.5 For every positive integer k,

$$\text{TR2-NFA}(k) \subset \text{TR2-NFA}(k+1).$$

Theorem 3.5 □

For the general case we were not able to prove such a strong hierarchy.

Theorem 3.6

$$\text{2-NFA}(n) - \text{2NFA}(o(n/\log_2 n)) \neq \varnothing.$$

Theorem 3.6 □

References

[AHY83] Aho, A.V., Hopcroft, J.E., Yannakakis, M.: On notions of information transfer in VLSI circuits. In: Proc. *15th Annual ACM STOCS*, ACM 1983, pp. 133–139.

[DHRS97] Ďuriš, P., Hromkovič, J., Rolim, J.D.P., Schnitger, G.: Las Vegas versus determinism for one-way communication complexity, finite automata and polynomial-time computations. In: Proc. *STACS '97, Lecture Notes in Computer Science* 1200, Springer 1997, pp. 117–128 (extended version submitted to Information and Computation).

[DKR94] Dietzelfelbinger, M., Kutylowski, M., Reischuk, R.: Exact lower bounds for computing Boolean functions on CREW PRAMs. *J. Computer System Sciences* 48 (1994), pp. 231-254.

[DT90] Diaz, J., Torán, J.: Classes of bounded nondeterminism. *Mathematical Systems Theory* 23 (1990), 21–32.

[Gi77] Gill, J.: Computational complexity of probabilistic Turing machines. *SIAM J. Comput.* 6 (1977), pp. 675–695.

[GKW90] Goldstine, J., Kintala, C.M.R., Wotschke, D.: On measuring nondeterminism in regular languages. *Information and Computation* 86 (1990), 179–194.

[HIT89] Hromkovič, J., Inoue, K., Takanami, I.: Lower bounds for language recognition on two-dimensional alternating multihead machines. *ICSS* 38 (1989), 431–451.

[Hr97] Hromkovič, J.: *Communication Complexity and Parallel Computing.* Springer-Verlag 1997.

[HSa99] Hromkovič, J., Sauerhoff, M.: Communication with restricted nondeterminism and applications to branching program complexity. Unpublished manuscript, RWTH Aachen 1999.

[HS96] Hromkovič, J., Schnitger, G.: Nondeterministic communication with a limited number of advice bits. In: *Proc. STOC'96, ACM* 1996, 551–560.

[HS99] Hromkovič, J., Schnitger, G.: On the power of Las Vegas II, Two-way finite automata. In: *Proc. ICALP'99*, *Lecture Notes in Computer Science* 1644, Springer 1999, 433–443.

[IITT82] Ito, A., Inoue, K., Takanami, I., Taniguchi, H.: Two-dimensional alternating Turing machines with only universal states. *Inform. & Control 55* (1982), 193–221.

[KN97] Kushilevitz, E., Nisan, N.: *Communication Complexity*, Cambridge University Press 1997.

[KW80] Kintala, C.M.R., Wotschke, D.: Amount of nondeterminism in finite automata. *Acta Informatica* 13 (1980), 199–204.

[MS82] Mehlhorn, K., Schmidt, E.: Las Vegas is better than determinism in VLSI and distributed computing. In: Proc. *14th ACM STOC'82*, ACM 1982, pp. 330–337.

[MS97] Macarie, I.I., Seiferas, J.I.: Strong equivalence of nondeterministic and randomized space-bounded computations. Manuscript.

[PS84] Papadimitrou, Ch., Sipser, M.: Communication complexity. *J. Computer System Sciences* 28, pp. 260–269

[Sa97] Sauerhoff, M.: On nondeterminism versus randomness for read-once branching programs. *Electronic Colloquium on Computational Complexity*, TR 97 - 030.

[Sa99] Sauerhoff, M.: On the size of randomized OBDDs and read-once branching programs for k-stable functions. In: Proc. *STACS '99*, *Lecture Notes in Computer Science*, to appear.

[Ya79] Yao, A.C.: Some complexity questions related to distributed computing. In: Proc. *11th ACM STOC, ACM* 1979, pp. 209–213.

A The Proof of Theorem 3.2

Let

$$L = \{\ y \in \{0,1,2\}^{(2)} \mid \exists n, j, l : l_1(y) = l_2(y) = n \geq 2, 1 \leq j \leq n-1,$$
$$0 \leq l \leq n-3, \text{ the row } j \text{ of } y \text{ is } 10^l 20^{n-l-2},$$
$$\text{every row above the } j\text{-th row is } 10^{n-2}1, \text{ and}$$
$$[(n-j-1 \leq l \wedge y(n,n) = 1) \vee (n-j-1 \leq n-l-3 \wedge y(n,n) = 0)]\}.$$

Note that the inequality $n-j-1 \leq l$ [the inequality $n-j-1 \leq n-l-2$] means that the symbol 2 in the j-th row of y is on or below the secondary [the main] diagonal of y.

We divide the proof of Theorem 3.2 into two parts: Theorem A.1 and Theorem A.2. Especially the proof of the lower bound in Theorem A.2 is of interest, because it provides a very careful analysis of the behaviour of tr2-dfa's.

Theorem A.1 L can be recognized by a three-way Las Vegas finite automaton.

Theorem A.1 □

Proof of Theorem A.1: There is a three-way Las Vegas finite automaton B recognizing L as follows. Let y be any input from $\{0,1,2\}^{(2)}$ with $l_1(y) = l_2(y) = n$. B first traverses y by row-by-row manner checking the structure of the scanned rows and looking for the first row containing the symbol 2. (B rejects y if it detects any error during that first phase.) Then B chooses (with probability 1/2) which of the two inequalities $n-j-1 \leq l$ or $n-j-1 \leq n-l-3$ to verify. B is able to verify any of these inequalities, since the distance from the symbol 2 of the row j to the fist column of y, to the last column of y, to the last row of y is l, $n-l-3$ and $n-j-1$, respectively. Finally, B is able to accept or reject y when B verifies the first (the second) inequality mentioned above and $y(n,n) = 1$ ($y(n,n) = 0$). Note that B rejects y also if $y(n,n) = 2$.

Theorem A.1 ■

Theorem A.2 L cannot be accepted by any three-way deterministic finite automaton.

Theorem A.2 □

Proof of Theorem A.2: Suppose to the contrary that there is such automaton A with k states recognizing L. W.l.o.g. we may assume that A visits each row of each recognized input. Let y be any input from $\{0,1,2\}^{(2)}$. A **cell** (i,j) of y is the entry of y in the row i and in the column j of y. A **configuration** of A is a triple (q,i,j), where q is the state of A, and i and j are nonnegative integers. We will say that A is in a configuration (q,i,j) on an input, if A is in the state q and its head scans the i-th row and the j-th column of the input. Let $n = k!(2k^2(k+2)) + 2k(k+2)$. Let x be an input with $2n+3$ rows and $2n+3$ columns such that the row $n+2$ is $10^n 20^n 1$ and every other row is $10^{2n+1}1$. Clearly, $x \in L$.

First we present an idea of the proof based on the following observation. If A recognizes the input x of very simple structure and x is large enough (with respect to the number

of states of A), then A works in cycles very often. This will enable us to fool A by relocating the symbol 2 to a suitable cell so that A is not able to recognize this relocation, since A is not able to count the number of performed cycles. More particularly, our aim is to construct an other input z (from x) by relocating the symbol 2 to a suitable cell so that both inputs x and z are either accepted or rejected by A. If our construction yields $z \notin L$, then we have done, since $x \in L$, and hence, A is not able to recognize L correctly. If $z \in L$, then our construction will still guarantee an existence of two inputs $x' \in L$ and $z' \notin L$ that are both either accepted or rejected by A, and hence, A is not able to recognize L correctly. To prove that both inputs x and z are either accepted or rejected by A, we will proceed as follows. Assume that during the computation of A on x, A enters the row of x with the symbol 2 in a column j_0 being in a state p_0 and then it enters a cycle (at a configuration c_u) after leaving this row. (Such behaviour of A on x is discussed in Case 2 below and it is more interesting than an other possible behaviour of A on x discussed in Case 1 below.) We will see that during the computation of A on z, A enters the row of z with the symbol 2 in the same column j_0 being in the same state p_0, then it enters the same cycle (as A does on x) after leaving this row and then it reaches the same configuration c_u after performing the cycle as many times as necessary. We will also see that the rest of the computation of A on x and on z (starting by c_u) is the same. Hence, both inputs x and z are either accepted or rejected by A.

The structure of the proof is as follows. First we prove two lemmas and a corollary needed for analyzing the computation of A on x. Then we construct the input z mentioned above and complete the proof.

Lemma A.3 Let a, b and j be any integers with $1 \leq a, b, j \leq 2n+3$ and $b - a = k$. Let y be any input with $2n+3$ rows and $2n+3$ columns such that all cells (j, i) of y with $a \leq i \leq b$ contain the symbol 0. Assume that there are two steps $t < t'$ of the computation of A on y such that A scans only the cells (j, i) with $a \leq i \leq b$ during the computation of A on y from t to t', A scans the cell (j, a) at t, and A scans the cell (j, b) at t'. Then there is an integer h, $0 \leq h \leq k$, and two steps τ, τ' with $t \leq \tau < \tau' \leq t'$ such that A is in the same state and its head is relocated h cells to the right (without leaving the row j) just after performing every $\tau' - \tau$ steps (starting at τ) of the computation of A on y (i.e. A enters a cycle at the step τ and then A works in this cycle) until A enters a cell (of the row j) containing a symbol different from 0 occurred to the right from the cell (j, b). Lemma A.3 □

Remark A.1 A similar lemma holds also for $a - b = k$. In such case, the text "to the right" of Lemma A.3 is replaced by "to the left" and, of course, "$a \leq i \leq b$" is replaced by "$b \leq i \leq a$".

Proof of Lemma A.3: Let τ_k be the earliest step after t at which A scans the cell $(j, a+k)$ and for $i = 0, 1, 2, \ldots, k-1$, let τ_i be the latest step before τ_k at which A scans the cell $(j, a+i)$. Since A has k states, there are two steps $\tau_r < \tau_s$ with $0 \leq r < s \leq k$ such that A is in the same state at τ_r and also at τ_s. One can observe that τ_r and τ_s are our desired steps τ and τ', since A scans only the cells with the symbol 0 during the steps from τ_r to τ_s and A is a deterministic automaton that cannot change

its behaviour while it scans the cells with the same symbol. Note that $h = s - r$.

Lemma A.3 ■

Corollary A.4 For $i = 1, 2, 3, \ldots, n+1$, let (i, m_i) be the cell scanned by A during the computation on x just before leaving the row i. Then $m_i \leq k+1$ or $2n+3-m_i \leq k+1$ for $i = 1, 2, 3, \ldots, n+1$.

Corollary A.4 □

Proof of Corollary A.4: Suppose to the contrary that $m_i \geq k+2$ and $2n+3-m_i \geq k+2$ for some i. Clearly, A has to accept $x \in L$. Hence, A has to visit both cells $(i, 1)$ and $(i, 2n+3)$ of x containing 1, since otherwise A would accept also an input not belonging to L obtained from x by replacing the symbol 1 in the non-visited cell $(i, 1)$ or $(i, 2n+3)$ by the symbol 0, a contradiction. Let t_1 be the latest step at which A scans the cell $(i, 1)$ or $(i, 2n+3)$. We present here a proof only for the case when A scans the cell $(i, 1)$ at t_1, i.e. A cannot visit the cell $(i, 2n+3)$ after t_1. (The proof for the second case is similar.) Thus, A has to visit (sometimes after t_1) every cell (i, j) with $2 \leq j \leq m_i$ containing 0. Let t_{k+2} be the earliest step after t_1 at which A scans the cell $(i, k+2)$, and let t_2 be the latest step before t_{k+2} at which A scans the cell $(i, 2)$. By Lemma A.3 (applied to the steps t_2 and t_{k+2}), A enters a cycle (sometimes between the steps t_2 and t_{k+2}) and it works in the cycle until it enters the cell $(i, 2n+3)$ containing 1. But it contradicts the assumption above that A cannot visit the cell $(i, 2n+3)$ after t_1.

Corollary A.4 ■

Let σ be the computation of A on x. Let t_0 be a step of σ at which A enters the row $n+2$ (containing 2). For $i = 0, 1, 2, \ldots$, let t_{i+1} be the earliest step of σ after t_i (if there exists) at which A scans the cell $(n+2, n+2)$ with the symbol 2 [at which A scans either the cell $(n+2, 1)$ or $(n+2, 2n+3)$ with the symbol 1] if i is even [odd]. Since A has to leave the row $n+2$ sometimes, there are only finitely many t_i's. Let $t_0, t_1, \ldots, t_d$ be all such t_i's. For $i = 0, 1, 2, \ldots, d$, let $(p_i, n+2, j_i)$ be the configuration of A on x at the step t_i.

Lemma A.5 Let d, t_i's, p_i's and j_i's be as above. Let p be any integer with $n - k!|p| \geq 2k+2$ and let y be an input identical to x with the exception that its cell $(n+2, n+2+k!p)$ contains 2 and its cell $(n+2, n+2)$ contains 0. Then (i) hold for every even $i < d$ and (ii) holds for every odd $i < d$.

(i) There is a computation path of A on y from the configuration $(p_i, n+2, j_i)$ to the configuration $(p_{i+1}, n+2, n+2+k!p)$.

(ii) There is a computation path of A on y from the configuration $(p_i, n+2, n+2+k!p)$ to the configuration $(p_{i+1}, n+2, j_{i+1})$.

Lemma A.5 □

Proof of Lemma A.5:

Proof of (i) : By the assumption above Lemma A.5, $j_i = 1$ or $j_i = 2n+3$ for even $i > 0$. By Corollary A.4, $j_0 \leq k+2$ or $2n+3-j_0 \leq k+2$, since A can change its head

position at most one cell (vertically and horizontally) during one step when A moves its head from a row to the next row. Hence $j_i \leq k+2$ or $2n+3-j_i \leq k+2$ for every even i, $0 \leq i \leq d-1$. Assume that $j_i \leq k+2$. (The proof for $2n+3-j_i \leq k+2$ is similar.) By the assumption above Lemma A.5, A has to traverse from the cell $(n+2, j_i)$ to the cell $(n+2, n+2)$ containing 2 (and hence from the cell $(n+2, j_i+1)$ to the cell $(n+2, j_i+k+1)$) during the steps from t_i to t_{i+1} of the computation on x. All cells $(n+2, j_i+l)$ with $1 \leq l \leq k+1$ contains 0. Let t be the earliest step after t_i at which A scans the cell $(n+2, j_i+k+1)$ and let t' be the latest step before t at which A scans the cell $(n+2, j_i+1)$. By Lemma A.3 (applied to the steps t' and t), A enters a cycle (sometimes between the steps t' and t) and then A works in that cycle until it reaches the cell $(n+2, n+2)$ containing 2. (Note that all cells $(n+2, j_i+l)$ with $1 \leq l \leq n+1-j_i$ contain 0.) Behaviour of A on x and on y is identical during $t-t_i$ steps starting by the configuration $(p_i, n+2, j_i)$, since A scans only the cells $(n+2, l)$ with $1 \leq l \leq j_i+k+1$ on x during these steps and both inputs x and y contain the same string 10^{j_i+k} in these cells. Hence, A enters the same cycle on y by the same way during these steps as A does on x, and clearly, then A works in that cycle on y until it reaches the cell $(n+2, n+2+k!p)$ containing 2. Since the distance between the cells containing the symbol 2 of x and y is $k!p$, we get the desired computation path of A on y mentioned above in (i) by inserting if $p > 0$ or by deleting if $p < 0$ $k!|p|/h$ cycles into a suitable point of the computation path of A on x from the step t_i to the step t_{i+1}, where h is the value from Lemma A.3 corresponding to the steps t' and t.

Proof of (ii) : By the assumption above Lemma A.5, $j_i = n+2$, and $j_{i+1} = 1$ or $j_{i+1} = 2n+3$ for odd i. Thus, A has to traverse from the cell $(n+2, n+2)$ containing 2 to the cell $(n+2, 1)$ or $(n+2, 2n+3)$ containing 1 (and hence from the cell $(n+2, n+2)$ to the cell $(n+2, n-k+1)$ or $(n+2, n+k+3)$) during the steps from t_i to t_{i+1} of the computation on x. Let t be the earliest step after t_i at which A scans the cell $(n+2, n-k+1)$ or $(n+2, n+k+3)$. Assume that A scans the cell $(n+2, n-k+1)$ at t. (The proof is similar if A scans the cell $(n+2, n+k+3)$ at t.) Let t' be the latest step before t at which A scans the cell $(n+2, n+1)$. By Lemma A.3 (formulated for $a > b$, see Remark A.1, and applied to the steps t' and t), A enters a cycle (sometimes between the steps t' and t) and then A works in that cycle until it reaches the cell $(n+2, 1)$ containing 1. (Note that all cells $(n+2, l)$ with $2 \leq l \leq n+1$ contain 0.) Behaviour of A during $t-t_i$ steps starting when A scans the cell containing 2 being in the state p_i is the same on x and also on y, since on x, A scans only the cells $(n+2, l)$ with $n-k+1 \leq l \leq n+k+2$ containing the string $w = 0^{k+1}20^k$ during these steps, and y contains the same string w in the cells $(n+2, l)$ with $n-k+1+k!p \leq l \leq n+k+2+k!p$. Thus, A enters the same cycle on y by the same way during these steps as A does on x, and clearly, then A works in that cycle on y until it reaches the cell $(n+2, 1)$ containing 1. Now one can observe that this is the desired computation path of A on y mentioned in (ii), since the distance between the cells containing the symbol 2 of x and of y is $k!p$, and hence, the computation path of A on x from t_i to t_{i+1} differs from the computation path of A on y mentioned above only in the number of performed cycles. Note that this difference is $k!p/h$ cycles, where h is the value from Lemma A.3 corresponding to the step t' and t.

Lemma A.5 ■

Now we are ready to complete the proof of the theorem. Let d, t_i's, p_i's and j_i's be from Lemma A.5. There are two cases to be considered.

Case 1. d is even, i.e. $j_d \leq k+2$ or $2n+3-j_d \leq k+2$, since if $d > 0$ then $j_d = 1$ or $j_d = 2n+3$, and if $d = 0$ then $j_0 \leq k+2$ or $2n+3-j_0 \leq k+2$ (see the beginning of the proof of (i) of Lemma A.5). Assume that $j_d \leq k+2$. (The proof for $2n+3-j_d \leq k+2$ is similar.) One can show (by the same way as in the proof of (i) of Lemma A.5) that if A visited the cell $(n+2, j_d+k+1)$ of x after t_d (i.e. if A traversed from the cell $(n+2, j_d+1)$ to the cell $(n+2, j_d+k+1)$ after t_d), then A would enter a cycle (sometimes after t_d) and then A would work in that cycle until it would reach the cell $(n+2, n+2)$ of x containing 2. But this would contradict the assumption above Lemma A.5, that t_d is the latest selected t_i. Hence A cannot visit the cell $(n+2, j_d+k+1)$ after t_d, i.e. A can scan (among the cells of the row $n+2$ of x) only the cells $(n+2, l)$ with $1 \leq l \leq j_d+k$ after t_d. Let z be an input that is identical to x with the exception that its cell $(n+2, n+2)$ contains 0, and its cell $(n+2, n+2-k!)$ contains 2. Clearly, $z \notin L$. Let us consider the computation of A on z. A enters the row $n+2$ of z by the same way as A does on x (i.e. in the configuration $(p_0, n+2, j_0)$), since the first $n+1$ rows of x and z are identical. By the assumption above Lemma A.5 there is a computation path of A on x from the configuration $(p_0, n+2, j_0)$ to the configuration $(p_d, n+2, j_d)$. By (i) and (ii) of Lemma A.5, there is such computation path of A on z, too. The rest of the computation of A (starting by the configuration $(p_d, n+2, j_d)$) is the same on x and also on z, since on x, A can scan (among the cells of the row $n+2$) only the cells $(n+2, l)$ with $1 \leq l \leq j_d+k \leq 2k+2$ after t_d (see above), these cells of x and z contain the same string 10^{2k+1}, and x and z agree on the last $n+1$ rows. Therefore A either accepts or rejects both inputs $x \in L$ and $z \notin L$, a contradiction.

Case 2. d is odd, i.e. $j_d = n+2$. Hence A is in the state p_d and scans the cell $(n+2, n+2)$ containing 2 at the step t_d of the computation on x. To construct our desired input z, we have to determine three integers r, l, m by analyzing the computation of A on x.

We determine r as follows. Let m_i's be from Corollary A.4, and for $i = 1, 2, \ldots, n+1$, let s_i be the state at which is A just before leaving the row i of x. By Corollary A.4 and by the fact that A has k states, there are two indices i, j, $1 \leq i < j \leq 2k(k+2)+1 < n+1$ with $s_i = s_j$ and $m_i = m_j$. It means that the sequence $\alpha = (s_i, m_i), (s_{i+1}, m_{i+1}), \ldots, (s_{n+1}, m_{n+1})$ is a periodic one with a period $r \leq 2k(k+2)$, since A is a deterministic automaton and the first $n+1$ rows of x contain the same string $10^{2n+1}1$.

Now let us determine l and m. Let R be a rectangle area of x with $4k+3$ columns and $k+2$ rows situated in x so that the symbol 2 of x is in the middle of the first row of R (see Fig. 1 (a)). Note that all cells of R contain 0 with the exception for the one cell containing 2. One can show (by the same way as in the proof of (ii) of Lemma A.5) that if A visited the cell $(n+2, n-k+1)$ or $(n+2, n+k+3)$ of x after t_d, then A would enter a cycle (sometimes after t_d) and then A would work in that cycle until it would reach the cell $(n+2, 1)$ or $(n+2, 2n+3)$ containing 1. But this would contradict the assumption above Lemma A.5 that t_d is the latest selected t_i. Hence A can visit (among the cells of the row $n+2$ of x) only the cells $(n+2, j)$ with

$n-k+2 \leq j \leq n+k+2$ after t_d. All these cells occur in the first row of R at most k cells to the left and at most k cells to the right from the cell containing 2. Assume that A leaves the row $n+2$ of x (containing the first row of R) at a step $t_d+\tau$ for some $\tau \geq 0$. Since A can visit at most $k+1$ cells during any $k+1$ steps, all cells visited by A during the steps from t_d to $t_d+\tau+k+1$ of the computation on x belong to R. For $i = 0, 1, 2, \ldots, \tau+k+1$, let $c_i = (e_i, g_i, h_i)$ be the configuration of A at the step t_d+i of the computation of A on x. Since A has k states, there are two configurations c_u and c_v with $\tau+1 \leq u < v \leq \tau+k+1$ and $e_u = e_v$. Let $l = g_v - g_u$ and $m = h_v - h_u$. (The values c_u, c_v, l and m are illustrated in Fig. 1(a).) Clearly, $0 \leq l \leq k$ and $0 \leq |m| \leq k$.

Now we are ready to construct the input z. The input z is identical to x with the exception that the symbol 2 of z is in the cell $(n+2-k!rl, n+2+k!rm)$ and the cell $(n+2, n+2)$ of z contains 0. To complete the proof of Case 2, first we have to show that (a) and (b) hold.

(a) There is a step t' of the computation of A on z at which A is in the state p_d and scans the symbol 2 of z (i.e. A is in the configuration $(p_d, n+2-k!rl, n+2+k!rm)$). (Note that p_d is from Lemma A.5.)

(b) There is a step of the computation of A on z at which A is in the configuration c_u.

Proof of (a) : By periodicity of the sequence α above, $s_{n+1-k!rl} = s_{n+1}$ and $m_{n+1-k!rl} = m_{n+1}$, i.e. A leaves the row $n+1-k!rl$ and $n+1$ of x by the same way. Therefore A enters the rows $n+2-k!rl$ and $n+2$ of x by the same way, too. But it means that A is in the configuration $(p_0, n+2-k!rl, j_0)$ just after entering the row $n+2-k!rl$ of x, since by the assumptions above Lemma A.5, A is in the configuration $(p_0, n+2, j_0)$ just after entering the row $n+2$ of x. Since x and z agree on the first $n+1-k!rl$ rows, A is in the configuration $(p_0, n+2-k!rl, j_0)$ just after entering the row $n+2-k!rl$ of z, too. Let y be an input that is identical to x with the exception that the cell $(n+2, n+2+k!rm)$ of y contains 2 and the cell $(n+2, n+2)$ of y contains 0. By (i) and (ii) of Lemma A.5, there is a computation path of A on y from the configuration $(p_0, n+2, j_0)$ to the configuration $(p_d, n+2, n+2+k!rm)$. (Recall that d is odd in Case 2.) This yields that there is also a computation path of A on z from the configuration $(p_0, n+2-k!rl, j_0)$ to the configuration $(p_d, n+2-k!rl, n+2+k!rm)$, since the row $n+2$ of y and the row $n+2-k!rl$ of z contain the same string. Now we have done, since during the computation of A on z, A first enters the configuration $(p_0, n+2-k!rl, j_0)$ and then A enters the configuration $(p_d, n+2-k!rl, n+2+k!rm)$, (see above).

Proof of (b) : Let t' be the step from (a). For $i = 0, 1, 2, \ldots, \tau+k+1$, let $c'_i = (e'_i, g'_i, h'_i)$ be the configuration of A on z at the step $t'+i$ of the computation of A on z. Let R' be an analogy of R for the input z. Since all cells visited by A during the steps from t_d to $t_d+\tau+k+1$ of the computation on x belong to R (see above), and corresponding cells of R and R' contain the same symbols, and A is in the state p_d and scans the symbol 2 at the step t_d of the computation on x (see the assumptions of Case 2 above) and also at the step t' of the computation on z (see (a)

above), the behaviour of A on z during the steps from t' to $t'+\tau+k+1$ is the same as the behaviour of A on x during the steps from t_d to $t_d+\tau+k+1$ (see Fig. 1(b)), i.e. $e'_i = e_i$, $g'_i = g_i - k!rl$ and $h'_i = h_i + k!rm$ for $i = 0, 1, 2, \ldots, \tau+k+1$. Therefore A leaves the first row of R' (containing 2) at the step $t'+\tau$ of the computation on z, and A scans only cells of R' (below the first row of R') containing only 0 during the steps from $t'+\tau+1$ to $t'+\tau+k+1$ of the computation on z. Consequently, A scans only cells (of R') containing 0 during the steps from $t'+u$ to $t'+v$. The facts above yield that after performing $v-u$ steps of the computation of A on z (starting by the step $t'+u$), A is in the state e_u (since $e'_u = e_u = e_v = e'_v$, see above) and its head is relocated l cells down and m cells to the right if $m \geq 0$ or $-m$ cells to the left if $m < 0$ (since $g_v - g_u = l = g'_v - g'_u$ and $h_v - h_u = m = h'_v - h'_u$, see above). But the same holds also for every next $v-u$ steps of the computation of A on z (i.e. A works in a cycle with $v-u$ steps) while A scans the cells containing 0, since A is a deterministic automaton and it cannot change its behaviour while it scans only cells with the same symbol. Therefore A reaches the configuration $c_u = (e_u, g_u, h_u)$ on the input z after performing the cycle $k!r$ times, i.e. at the step $t'+u+k!r(v-u)$, (see Fig. 1(b)). This complete the proof of (b).

Now we complete the proof of Case 2 as follows. By (b), A enters the same configuration c_u during the computation on x and on z, too. The rest of the computation of A on x and on z (starting by c_u) is the same, since x and z agree on the last $n+1$ rows containing the cell scanned by A in the configuration c_u. Therefore both inputs x and z are either accepted or rejected by A. But it means that A is not able to recognize L correctly if $m < 0$, since in such case, $x \in L$ and $z \notin L$. Finally, we proceed as follows for $m \geq 0$. Let x', z' be two inputs identical to x and z, respectively, with an exception that the cell $(2n+3, 2n+3)$ of x' and z' contains 0. A reaches c_u on x' by the same way as it does on x, since x and x' agree on all rows above the last one and the cell scanned by A at the configuration c_u is also above the last row (it belongs to R). Clearly, the same holds for the inputs z and z'. Moreover, the rest of the computation of A on x' and z' (starting by c_u) is also the same, since x' and z' agree on the last $n+1$ rows containing the cell scanned by A in the configuration c_u. Therefore, both inputs x' and z' are either accepted or rejected by A. But it means that A is not able to recognize L correctly, since $x' \in L$, $z' \notin L$. Theorem A.2 ■

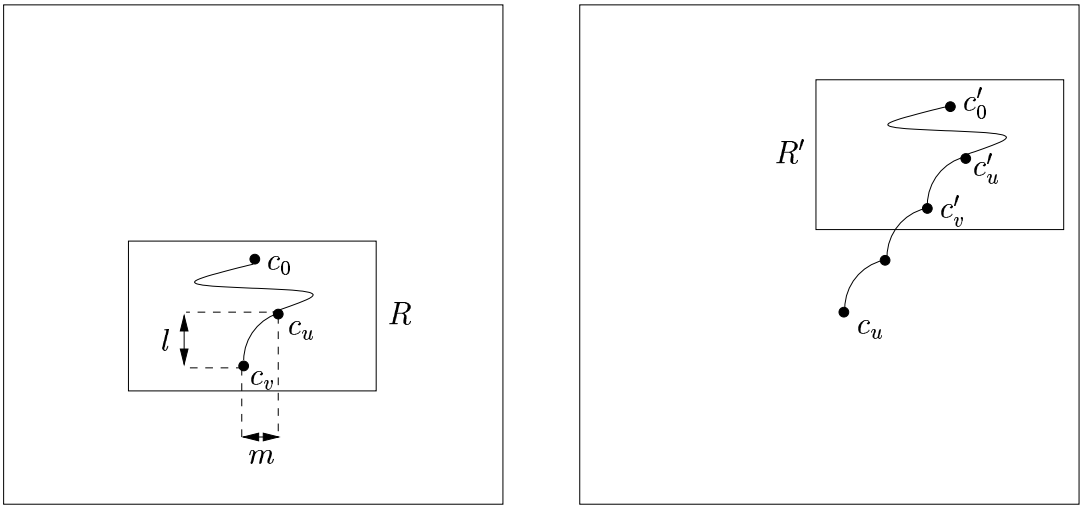
B Nondeterministic Hierarchies

B.1 Three-way case

We first consider a hierarchy based on the non-constant number of computation paths for three-way machines, and prove Theorem B.1.

Theorem B.1 For any $f(n) = o(\log n)$, any $g(n) \in \{(\log\log n)^k | k \text{ is a positive integer}\} \cup \{r^{\log\log n} | r \text{ is a positive integer}\}$, and any $g'(n) = o(g(n))$,

$$\text{TR2-NTM}(\log\log n, g(n)) - \text{TR2-NTM}(f(n), g'(n)) \neq \varnothing.$$



(a) Path from c_0 to c_v on x (b) Path from c'_0 to c_u on z

Figure 1:

Theorem B.1 □

Proof of Theorem B.1: We prove the theorem for $g(n) = (\log\log n)^k$. The proofs for the other cases are similar. Let

$$\begin{aligned} T_1 \;=\; & \{x \in \{0,1,\#\}^{(2)} | \exists n \geq 1 [l_1(x) = l_2(x) = |B(1)\#B(2)\#\ldots\#B(n)| \\ & \wedge \text{ (the first row of } x \text{ is } B(1)\#B(2)\#\ldots\#B(n)) \\ & \wedge \text{ (the second and the third rows of } x \text{ have exactly } (\log\log n)^k \text{ '1's)} \\ & \wedge \text{ (the second and third rows of } x \text{ are not the same)]}\}, \end{aligned}$$

where for each i $(1 \leq i \leq n)$, $B(i)$ denotes the binary representation (with no leading zeros) of the integer i.

T_1 is accepted by a $\log\log n$ space-bounded and $(\log\log n)^k$ path-bounded tr2-ntm M which acts as follows. Let x be an input tape to M. M first checks whether the first row of x is $B(1)\#B(2)\#\ldots\#B(n)$ for some $n \geq 1$. This check can be done deterministically by using $\log\log n$ cells of the storage tape. If this check is successful, then M checks whether the second row of x has exactly $(\log\log n)^k$ '1's. This check can also be deterministically done by using $\log\log n$ cells of the storage tape. If this check is successful, then each time M meets the symbol '1' (except the rightmost '1') on the second row of x, M nondeterministically chooses one of the following two actions:

1. One action is to go down one cell, and to pick up the symbol under the input head. If this symbol is not 1, and the third row of x has exactly $(\log\log n)^k$ '1's, then M enters an accepting state. Otherwise, M halts by entering a rejecting state.

2. The other action is to go to the right until M meets the next '1'.

When M meets the rightmost '1' on the second row of x, M acts just like 1. above.

It will be obvious that M accepts T_1.

We below show that $T_1 \notin \text{TR2-NTM}(f(n), g'(n))$, where $f(n) = o(\log n)$ and $g'(n) = o((\log\log n)^k)$.

Suppose to the contrary that there is an $f(n)$ space-bounded and $g'(n)$ path-bounded tr2-ntm M' accepting T_1. For each $n \geq 1$, let

$$\begin{aligned} V(n) &= \{x \in \{0,1,\#\}^{(2)} | l_1(x) = l_2(x) = |B(1)\#B(2)\#\ldots\#B(n)| \\ &\wedge \text{ (the first row of } x \text{ is } B(1)\#B(2)\#\ldots\#B(n)) \\ &\wedge (x[(2,1),(2,l_2(x))] = x[(3,1),(3,l_2(x))] \in \{0,1\}^{(2)}) \\ &\wedge (x[(2,1),(2,l_2(x))] \text{ has exactly } (\log\log n)^k \text{ '1's }) \\ &\wedge (x[(4,1),(l_1(x),l_2(x))] \in \{0\}^{(2)})\}. \end{aligned}$$

Note that each x in $V(n)$ is not in T_1, and so there doesn't exist an accepting computation path on M' on x.

For each x in $V(n)$, let $conf(x)$ be the set of possible configurations just after the input head of M' reached the third row of x (when x is presented to M').

Since, for each x in $V(n)$, there are at most $g'(r(n))$ computation paths of M' on x (where $r(n) = |B(1)\#B(2)\#\ldots\#B(n)| = O(n\log n)$), it follows that $|conf(x)| \leq g'(r(n))$. Letting $c(n)$ be the number of possible configurations just after the input head of M' reached the third rows of tapes in $V(n)$, we have

$$c(n) = O(r(n)t^{f(r(n))}) = O((n\log n)t^{f(r(n))})$$

for some constant t. For each $n \geq 1$, let

$$W(n) = \{conf(x) | x \in V(n)\}.$$

It follows that

$$|W(n)| \leq \sum_{i=0}^{g'(r(n))} \binom{c(n)}{i}.$$

Clearly,

$$|V(n)| = \binom{r(n)}{(\log\log n)^k}.$$

Since $\lim_{n\to\infty} f(n)/\log n = 0$ (by assumption), it follows that

$$\lim_{n\to\infty} f(r(n))/\log r(n) = \lim_{n\to\infty} f(r(n))/(\log n + \log\log n) = 0,$$

from which we have

$$\lim_{n\to\infty} f(r(n))/\log n = 0 \tag{1}$$

Further, since $\lim_{n\to\infty} g'(n)/(\log\log n)^k = 0$ (by assumption), it follows that

$$\lim_{n\to\infty} g'(r(n))/(\log\log r(n))^k = 0,$$

from which we have

$$\lim_{n\to\infty} g'(r(n))/(\log\log n)^k = 0 \qquad (2)$$

From formulas 1. and 2. above, we can easily derive that

$$|V(n)| > |W(n)|$$

for large n. Therefore, it follows that for large n there must be two different tapes x, y in $V(n)$ such that $conf(x) = conf(y)$.

Let u be the tape such that

$u[(1,1),(2,l_2(u))] = x[(1,1),(2,l_2(x))]$ and $u[(3,1),(l_1(u),l_2(u))] = y[(3,1),(l_1(y),l_2(y))]$.

From $conf(x) = conf(y)$ and from the fact that there does not exist any accepting computation path of M' on x and y, it follows that there does not exist any accepting computation path of M' on u, and thus u is not accepted by M', which contradicts the fact that u is in T_1. This completes the proof of "$T_1 \notin$ TR2-NTM$(f(n), g'(n))$".

Theorem B.1 ■

We next consider a hierarchy based on the constant number of computation paths for three-way machines. The following Theorem B.2 directly implies Theorem 3.5.

Theorem B.2 For any integer $k \geq 1$ and any $f(n) = o(\log n)$,

$$\text{TR2-NFA}(k+1) - \text{TR2-NTM}(f(n), k) \neq \varnothing.$$

Theorem B.2 □

Idea of the proof. For each integer $k \geq 1$, let $T(k) = \{x \in \{0,1\}^{(2)} | l_1(x) = l_2(x) \geq k \wedge$ (both the second and third rows of x have exactly k '1's) $\wedge$ (the second and third rows of x are not the same) $\}$.

Then, by using the same idea as in the proof of Theorem B.1, one can show that $T(k+1) \in \text{TR2-NFA}(k+1) - \text{TR2-NTM}(f(n), k)$.

Theorem B.2 ■

B.2 Four-way case

Next, we prove Theorem 3.6. To do it, we need some new notation an a technical lemma.

For each $n \geq 1$, each tape in $\{0,1\}^{n\times n}$ is called an ***n*-chunk**. Let M be a 2-ntm whose input alphabet is $\{0,1\}$. The number of the entrance points to an n-chunk x (or the exit points from x) for M is $4n$. We suppose that these entrance points (or exit points) are numbered $1, 2, \ldots, 4n$ in an appropriate way. Let $P(n) = \{1, 2, \ldots, 4n\}$ be the set of these entrance points (or exit points). For each non-negative integer l, let $S(l)$ be the set of possible storage states of M using at most l cells of the storage tape, where a

storage state of M is a combination of the (i) state of the finite control, (ii) contents of the storage tape, and (iii) position of the storage tape head within the nonblank portion of the storage tape. For each n-chunk x, and each non-negative integer l, let

$$M_{x,l} : P(n) \times S(l) \to 2^{P(n) \times S(l) \cup \{loop\}}$$

be a mapping which is defined as follows (where '*loop*' is a new symbol):

1. $(j,p) \in M_{(x,l)}(i,q) \iff$ when M enters the n-chunk x in storage state q and at point i, it may eventually exit from x in storage state p and at point j.
2. $loop \in M_{(x,l)}(i,q) \iff$ when M enters the n-chunk x in storage state q and at point i, it may not exit from x at all.

Let x, y be any two n-chunks. We say that x and y are **(M,l)-equivalent** if for any $(i,q) \in P(n) \times (l)$,

$$M_{(x,l)}(i,q) = M_{(y,l)}(i,q).$$

Clearly, (M,l)-equivalence is an equivalence relation on n-chunks, and we can easily see that the following lemma holds.

Lemma B.3 If, for any n-chunk x and any $(i,q) \in P(n) \times S(l)$,

$$|M_{(x,l)}(i,q)| \leq d,$$

where d is a positive integer, then there are at most

$$e(n,l,d) = \left\{ \sum_{i=1}^{d} \binom{4ns(l)+1}{i} \right\}^{4ns(l)}$$

(M,l)-equivalence classes of n-chunks, where $s(l) = |S(l)|$. Lemma B.3 □

We are now ready to prove the following theorem.

Theorem B.4 For any $f(n) = o(\log n)$ and any $g(n)$ such that $\log g(n) = o(\log n)$,

$$\text{2-NFA}(n) - \text{2-NTM}(f(n), g(n)) \neq \varnothing.$$

Theorem B.4 □

Proof of Theorem B.4: Consider the language $Mid = \{w \in \{0,1\}^{(2)} | \exists n \geq 0\ [l_1(w) = l_2(w) = 2n+1, w(n+1,n+1) = 1$ (i.e., the center symbol of w is '1') $]\}$. It is implicitly in [IITT82] that $Mid \in$ 2-NFA(n). Below, we show that $Mid \notin$ 2-NTM$(f(n), g(n))$ for $f(n)$ and $g(n)$ stated in the theorem. Suppose to the contrary that Mid is accepted by an $f(n)$ space-bounded and $g(n)$ path-bounded 2-ntm M. We assume without loss of generality that when M accepts an input w in Mid, it halts on the upper left-hand corner of w. For each $n \geq 0$, let $V(n) = \{0,1\}^{(2n+1)\times(2n+1)}$. We consider the case when

M reads inputs in $V(n)$. For each w in $V(n)$, M can use at most $f(2n+1)$ cells of the storage tape, and there are at most $g(2n+1)$ computation paths of M on w. Thus, from Lemma B.3 above, it follows that there are at most

$$e(n, f(2n+1), g(2n+1)) = \left\{ \sum_{i=1}^{g(2n+1)} \binom{4ns(f(2n+1))+1}{i} \right\}^{4ns(f(2n+1))}$$

$(M, f(2n+1))$-equivalence classes of n-chunks, where $s(f(2n+1)) = |S(f(2n+1))| = O(t^{f(2n+1)})$ for some constant t. We denote these $(M, f(2n+1))$-equivalence classes by $C_1, \ldots, C_{e(n,f(2n+1),g(2n+1))}$. Clearly, the total number of n-chunks is 2^{n^2}. Since $f(n) = o(\log n)$ and $\log g(n) = o(\log n)$ (by assumption), a simple calculation reveals that

$$e(n, f(2n+1), g(2n+1)) < 2^{n^2}$$

for large n. For such n, there must be some C_i $(1 \leq i \leq e(n, f(2n+1), g(2n+1)))$ such that $|C_i| \geq 2$. Let x, y be two different n-chunks in C_i and suppose without loss of generality that for some v_1, v_2 $(1 \leq v_1, v_2 \leq n)$, $x(v_1, v_2) = 1$ and $y(v_1, v_2) = 0$. We consider two tapes $w, w' \in V(n)$ satisfying the following two conditions:

(i) $w[(n - v_1 + 2, n - v_2 + 2), (2n - v_1 + 1, 2n - v_2 + 1)] = x$ and
$w'[(n - v_1 + 2, n - v_2 + 2), (2n - v_1 + 1, 2n - v_2 + 1)] = y$;

(ii) w and w' are the same except the segment described in (i).

As easily seen, w is in Mid, and so w is accepted by M. Since x and y are $(M, f(2n+1))$-equivalent, it follows that w' is also accepted by M, which contradicts to the fact that w' is not in T_2. This completes the proof of "$T_2 \notin$ 2-NTM$(f(n), g(n))$". Theorem B.4 ■

A straightforward modification of the calculation in the proof of Theorem B.4 provides the result of Theorem 3.6.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'