Computing and Informatics, Vol. 21, 2002, 617–624

A SHARP SEPARATION OF SUBLOGARITHMIC SPACE COMPLEXITY CLASSES*

Stanislav ŽÁK

Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Praha 8, Czech Republic
e-mail: stan@cs.cas.cz

Manuscript received 24 June 2002; revised 5 November 2002
Communicated by Juraj Hromkovič

Abstract. We present very sharp separation results for Turing machine sublogarithmic space complexity classes which are of the form: For any, arbitrarily slow growing, recursive nondecreasing and unbounded function s there is a $k \in N$ and an unary language L such that $L \in SPACE(s(n) + k) \setminus SPACE(s(n-1))$. For a binary L the supposition $\lim s = \infty$ is sufficient. The witness languages differ from each language from the lower classes on infinitely many words. We use so called demon (Turing) machines where the tape limit is given automatically without any construction. The results hold for deterministic and nondeterministic demon machines and also for alternating demon machines with a constant number of alternations, and with unlimited number of alternations. The sharpness of the results is ensured by using a very sensitive measure of space complexity of Turing computations which is defined as the amount of the tape required by the simulation (of the computation in question) on a fixed universal machine. As a proof tool we use a succint diagonalization method.

Keywords: Space complexity, separation, diagonalization

1 INTRODUCTION

One of the basic tasks of the theory of computational complexity is separation of complexity classes. For a given computational device and for a given complexity

* This research was supported by GA CR, Grant No. 201/02/1456.

measure which controls the computational resource in question it is necessary to find the smallest enlarging of the complexity bound which strengthens the computational power. In the 60's and 70's this question arose for classical Turing machines and for time and space complexity measures (e.g. [2, 12, 13, 14] — and newly [7]). Whenever a new computational device or a new complexity measure are defined it is quite natural (and necessary) to solve this question.

In the 90's such a situation arose in the theory of computations on Turing machines with sublogarithmic tape ([1, 3, 4, 5, 8, 9, 10, 15]). However, in the context of this theory there was a disagreable technical difficulty, namely the impossibility to construct the space bounds deeply below $\log n$. This difficulty was solved in a radical manner; in [11] the authors introduced so-called demon machines — a new type of Turing machines with sublogarithmic tape where the space bound is given in advance and gratis. The demon machines are treated also in [15, 6]. Due to this approach we can work with arbitrarily small limit functions and we can concentrate only on the computability within these bounds. Thus it is easier to prove some separation results also below log. Geffert solves this problem and among others in [6, 7] he proves some separation results of the classical form: If the separation condition $\lim s_2(n)/s_1(n) = \infty$ holds, then $SPACE(s_2(n)) \supset SPACE(s_1(n))$.

We want to prove some separation results with a finer separation condition. Of course, we must use a modified, more sensitive, space complexity measure which involves also the number of symbols which machines may write on their worktapes and, moreover, the amount of their finite control states. The complexity of a computation of a Turing machine is given by the amount of the tape required by its simulation on a fixed universal machine. This approach gives us the possibility to prove the results mentioned in the Abstract. Moreover, the fact that the results also hold for alternating demon machines is not without any interest since the situation with space complexity classes for alternating machines below log is quite different from that over log [1, 4, 5, 10, 15].

As a proof tool we use a general diagonalization method. Its ancestor — a relatively cumbersome construct — was used for separating space complexity classes in [16] and for separating complexity classes defined by the amount of information yielded by an oracle in [17, 18]. The final version of this diagonalization method is very succint and clear. It was used in [19] for time complexity classes and now for the first time it is used for space complexity. Due to its generality the presented proofs can be interpreted as the proofs for a number of various Turing machines (e.g., deterministic, nondeterministic, alternating Turing machines, Turing machines with an auxiliary pushdown store, and with an oracle).

2 THE BASIC COMPUTATIONAL MODEL

As a standard model (acceptor) of computation we shall consider nondeterministic Turing machine having a two-way read-only input tape and a separate semi-infinite two-way read-write worktape.

Definition 2.1 ([6])**.** For any function s, a demon s-tape bounded machine begins its computation with a special tape limit marker placed exactly $s(n)$ positions away from the initial position of the worktape, for every input of length n. The tape marker can be detected (and cannot be moved). The machine rejects if it ever tries to use more than $s(n)$ tape.

Alternating demon s-tape bounded machines may have their computation trees infinite since for small functions $s(n)$ below log the demon machines have difficulties to avoid cycles in their computations. For the evaluation of an infinite computation tree we effectively construct its reduced finite version. We cut each its branch at the moment when some configuration is repeated for the first time. The leaves of this kind are marked as rejecting. Then we evaluate the tree in the usual way.

We fix a universal machine U. On the input tape, U reads only the symbols $0, 1$ and the endmarkers. On the worktape, U works with $0, 1, b$ and the endmarkers. The programs of demon machines are well-structured strings of 0's and 1's. U starts its computation in the situation when the program of the simulated machine is placed in the leftmost part of the worktape. U has the property that for each demon $s(n)$-tape bounded machine M using on the worktape only the symbols $0, 1, b$ and the endmarkers, U simulates M on the tape of the length $s(n) + |p_M|$ where p_M is the program of M. (Moreover, for each $m \geq 4$ there is a constant k_m such that the U-simulation of the machines which use m worktape symbols increases the tape complexity only by the multiplicative konstant k_m).

For tape bounded deterministic and nondeterministic demon machines and also for alternating machines with a fixed number of alternations and for machines with unlimited number of alternations we define complexity classes as follows.

Definition 2.2. Let p be a program of a machine and s be a function. By $L_s(p)$ we mean the language of words x which are accepted by U when U starts with the program p in the leftmost part of its worktape and with its worktape endmarker on the position $s(|x|) + |p|$. Further we define $SPACE(s(n)) =_{df} \{L_s(p) | p$ is a program of a machine$\}$.

Let s, s_1 be functions, $s_1(n) < s(n)$. We say that s_1 is s-constructible iff there is a deterministic $s(n)$-tape bounded demon machine M such that on the inputs of length n M ends with 1 on the $s_1(n)$-th position (all other positions till $s(n) - 1$ are blank).

3 THE DIAGONALIZATION THEOREM

The principle of our diagonalization can be formulated without any notions concerning computability or complexity.

For languages L_1, L_2 we say that L_1 is equivalent to L_2 ($L_1 \sim L_2$) iff L_1, L_2 differ only on a finite number of words. For a class C of languages, by $\mathcal{E}(\mathcal{C})$ we mean the class $\{L' | (\exists L \in C)(L' \sim L)\}$.

Theorem 3.1 (The d-theorem, [19]). Let L be a language and let C be a class of languages indexed by a set S, $C = \{L_p : p \in S\}$. Let R be a language and let F be a mapping, $F : R \to S$, such that $(\forall p \in S)(\exists^{\infty} r \in R)(F(r) = p)$. Let z be a mapping, $z : R \to N$, such that for each $r \in R$, $z(r)$ satisfies the following two conditions: (a) $r1^{z(r)} \in L \leftrightarrow r \notin L_{F(r)}$, (b) $(\forall j, 0 \leq j < z(r))(r1^j \in L \leftrightarrow r1^{j+1} \in L_{F(r)})$. Then $L \notin \mathcal{E}(C)$.

Comment. In our application, C will be the complexity class to be diagonalized over, and S will be the set of programs, R the set of their codes. Our task is to construct a diagonalizer M which will accept a language L, $L \notin \mathcal{E}(\mathcal{C})$. The features of M are well described by the conditions (a),(b). On the input $r1^j$, M will derive the program $F(r)$ and then M will try whether the input is fully padded ($j = z(r)$; if so, then M will decide whether $r \in L_{F(r)}$ (cf. the condition (a)). For inputs which are not fully padded, M will simulate the program $F(r)$ on the input with one additional '1' (cf. the condition (b)). More details can be found in the proofs of theorems below.

This diagonalization can be compared with the classical one from the point of computational complexity. The classical diagonalization construct the desired L in such a way that $r \in L \leftrightarrow r \notin L_{F(r)}$. We know that the decision whether $r \in L_{F(r)}$ requires a large amount of computational sources (space, time, ...), especially in case of nondeterministic computations. In our diagonalization we use the fact that the complexity is defined in relation to the length of inputs. To decide whether $r \in L_{F(r)}$ we use very long inputs $r1^{z(r)}$; so the resulting complexity is very small and negligible. The largest consumption of computational sources is now concentrated in our condition b). Given an input of length n, we simulate a computation on the input of length $n+1$. Even in case of nondeterministic computations the respective increase of complexity is moderate.

Proof. By contradiction. Suppose $L \in \mathcal{E}(C)$. Hence $L \sim L_p$ for some $p \in S$. Moreover, there is an $r \in R$ such that $F(r) = p$ and the languages $L, L_{F(r)}$ $(=L_p)$ differ only on words shorter than r; in particular for each $j \in N, r1^j \in L_{F(r)}$ iff $r1^j \in L$. Hence by condition (b) $r \in L \leftrightarrow r1^{z(r)} \in L$, and then by condition (a) $r1^{z(r)} \in L \leftrightarrow r \notin L$. A contradiction. □

4 THE SEPARATION RESULTS

We apply the previous theorem to the complexity classes induced by deterministic and nondeterministic demon machines or by alternating demon machines with a fixed number of alternations or by machines with unlimited number of alternations. For each such type of machines the following two theorems hold.

Theorem 4.1. Let s be a recursive function, $\lim s(n) = \infty$. Then there is a constant $k \in N$ and a language $L \subseteq 1^+0^+1^*$ such that $L \in SPACE(s(n) + k) - \mathcal{E}$ $SPACE(s(n-1))$.

Proof. We shall apply our d-theorem. Let S be a recursive set of programs of the machines in question.We put $C =_{df} SPACE(s(n-1))$. For $p \in S$ we define $L_p = L_{s(n-1)}(p)$. Then $C = \{L_p | p \in S\}$. Further we define the set of program codes $R =_{df} \{1^k 0^l |\ |bin(k)| \leq s(k+l) - 1 \wedge bin(k) \in S \wedge (\forall j)(s(k+l) \leq s(k+l+j))\}$, and for $r = 1^k 0^l \in R$, $F(r) =_{df} bin(k)$ (where $bin(k)$ is the binary code of k). We see that $(\forall p \in S)(\exists^\infty r \in R)(F(r) = p)$.

Our diagonalizer M first checks whether the input is of the form $1^+0^+1^*$. If the input is of the form $1^k0^l1^j$ then M tries to construct $bin(k)$ on its worktape. If $bin(k) = p \in S$, then, using a recursive procedure, M tries to decide whether $1^k0^l \in L_p$. If M is able to decide whether $1^k0^l \in L_p$, not using more space than $s(k+l+j)-1$, it accepts iff $1^k0^l \notin L_p$. Otherwise M simulates p on $1^k0^l1^{j+1}$ (as U).

Let $r = 1^k0^l \in R$. We define $z(r) =_{df} \min\{j | on\ r1^j\ M$ can decide whether $r \in L_{F(r)}\}$. We see that $r1^{z(r)} \in L(M)$ iff $r \notin L_{F(r)}$ (the condition (a) of the d-theorem). Further, for $j < z(r)$ $r1^j \in L(M)$ iff $r1^{j+1} \in L_{s(n-1)}(F(r)) = L_{F(r)}$ (the condition (b) of the d-theorem). Hence, $L(M) \notin \mathcal{E}\ (C) = \mathcal{E}\ (SPACE(s(n-1)))$.

Since the universal machine U uses only the symbols $0, 1, b$ and an endmarker, we are able to construct our diagonalizer M in such a way that M uses only these symbols as well. Hence, the U-simulation of the segment of length $s(n)$ of the worktape of M requires $s(n) + |p_M|$ cells only. Therefore, $L \in SPACE(s(n) + k)$ for some $k \in N$. □

The proof for the case of unary witness language L is more complicated.

Theorem 4.2. Let s be a recursive function such that there is an s-constructible nondecreasing and unbounded function s_1. Then there is a constant $k \in N$ and a language $L \subseteq 1^+$ such that $L \in SPACE(s(n) + k) - \mathcal{E}\ SPACE(s(n-1))$.

Proof. We prepare the situation for an application of the d-theorem. Let S be a recursive set of programs of the machines in question, recognizing unary languages. For $p \in S$ we define $L_p =_{df} L_{s(n-1)}(p)$. We put $C =_{df} SPACE(s(n-1)) = \{L_p | p \in S\}$.

Our diagonalizer M will compute as follows. M first checks whether the input is an unary string. Then, on its worktape, M constructs the value of $s_1(n)$, that is, the corresponding position of the worktape is rewritten by the symbol "1".

Within the first $s_1(n) - 1$ cells of its worktape, M will perform an initial part of a recursive process P which we shall describe. P will give us the possibility to define R, F and z which we need for the application of the d-theorem.

P contains a generator of the programs $p_1, p_2, \ldots$ from S such that if $\{p_i\}$ is the generated sequence then $(\forall p \in S)(\exists^\infty i)(p_i = p)$.

P starts with the generation of p_1. At this moment some amount a_1 of the worktape has been used. Then P constructs $n_1 =_{df} \min\{n | s_1(n) - 1 \geq a_1\}$. Then P decides whether $1^{n_1} \in L_{p_1}$ or not.

If P has generated p_i, constructed 1^{n_i} and decided whether $1^{n_i} \in L_{p_i}$ then (on the non-used cells) P generates p_{i+1}. At this moment some amount a_{i+1} of the worktape has been used. Then P constructs
$n_{i+1} =_{df} \min\{n | s_1(n) - 1 \geq a_{i+1}\}$ and then P decides whether $1^{n_i+1} \in L_{p_{i+1}}$ or not.

Now, we define $r_i =_{df} 1^{n_i}$, $R =_{df} \{r_i | i \in N\}$, $F(r_i) =_{df} p_i$. Further, $z(r_i)$ is the first m such that $s_1(|r_i| + m) - 1$ cells suffices for P to generate p_i, to construct r_i and to decide whether $r_i \in L_{p_i}$.

We continue the description of M. When P is stopped because of the lack of space on the worktape, P has used all $s_1(n) - 1$ cells. Then "the result of P" denotes the last generated p_i and also the decision whether $r_i \in L_{p_i}$ if this decision is achieved. (Let $i(n)$ be the index of the result (p_i) of P when computing on the input word of length n. Since $s_1(n)$ is a nondecreasing function the sequence $\{i_n\}$ is nondecreasing, too.)

If this decision is achieved, then the input is of the form $r_i 1^j$, $j \geq z(r_i)$. M accepts iff $r_i \notin L_{p_i}$. Let L be the language accepted by M. We have $r_i 1^{z(r_i)} \in L \leftrightarrow r_i \notin L_{p_i} = L_{F(r_i)}$ — the condition (a) of the d-theorem is satisfied.

If the decision in question is not achieved, then M computes as the universal machine U on the input $1^{n+1} (= r_i 1^{j+1})$ with the program p_i on the leftmost part of the worktape. We have $r_i 1^j \in L \leftrightarrow r_i 1^{j+1} \in L_{p_i} = L_{F(r_i)}$ (the condition (b) of the d-theorem is satisfied). We have proven $L \notin \mathcal{E}\ SPACE(s(n-1))$.

Since the universal machine U uses only the symbols $0, 1, b$ and an endmarker, we are able to construct our diagonalizer M in such a way that M uses only $0, 1, b$, and the endmarker, too. Hence, the U-simulation of the segment of length $s(n)$ of the worktape of M requires $s(n) + |p_M|$ cells only. Therefore $L \in SPACE(s(n) + k)$ for some $k \in N$. □

Remark 1. It does not seem to be possible to convert the separation results into the hierarchy results of the form $SPACE(s(n) + k) \supset SPACE(s(n-1))$ since there are such recursive functions s that the sets $\{n | s(n) \neq s(n-1)\}$ are of very high complexities. This is a negative consequence of the elegance of the definition of the demon machines, and of their non-constructiveness. There is an open question whether there is a reasonable class of small recursive functions such that the result conversion mentioned above is possible.

Remark 2. We are also able to prove the hierarchy results for bounds over $\log(n)$. In this case we can use the concept of constructible bounds $s(n)$; the computation starts by a deterministic part using only $s(n)$ cells after which the endmarker is posed on $s(n)$-th cell of the worktape. If such a machine M is simulated on a universal machine U which has its own tape limit, the endmarker of M is simulated as the first 1 to the left from the endmarker of U. In case of such constructible bounds we are able to prove very sharp hierarchies. For linear and sublinear bounds, enlarging of the bound by an additive constant increases the computation power of the machines in question (cf. [16]).

Remark 3. Similar results can be proven for machines with one auxiliary pushdown store for both bounds below $\log n$ (demon bounds) and bounds over $\log n$

(constructible bounds). The structure of our proof remains unchanged. The recursive procedure — used for sufficiently padded inputs $r1^{z(r)}$ — may be powerful enough to decide whether $r \in L_{F(r)}$ also for machines with one auxiliary pushdown. The second branch of computation of our diagonalizer — the simulation — is easy also in case of machines with an auxiliary pushdown. Hence, our separation (and hierarchy) results remain valid also for machines with an auxiliary pushdown store.

Remark 4. Similar results can be proven for machines with a recursive oracle. Such a machine has a special — oracle — write-only tape and three special states — *QUERY*, *YES*, *NO*. If the machine enters the state *QUERY* then the next state is *YES* iff the word written on the oracle tape belongs to the oracle, and *NO* otherwise. After this action the oracle tape is erased. Our diagonalizer may be powerful enough to decide the recursive oracle when computing on the fully padded inputs. The second branch of its computation — the simulation — is easy. Hence, also for machines with an oracle our separation results remain valid.

Remark 5. The same proof structure is (probably) applicable for Turing machines with an oracle where not only the space but also the use of oracle are measured (and bounded with respect to the length of inputs) as the computational resources. E.g. the bounds on questions to the oracle can limit the number of questions or the sum of lengths of questions or the maximal length of questions. The necessity to measure two computational resources (space, oracle) would imply some rearrangements in the statement of the possible theorem and in its proof.

Acknowledgment

I thank V. Geffert for his helpful comments on an early version of the text.

REFERENCES

[1] von Braunmühl, B.—Gengler, R.—Rettinger, R.: The Alternation Hierarchy for Machines with Sublogarithmic Space is Infinite. Proc. of STACS'94, LNCS 775, Springer-Verlag, pp. 85–96, 1994.

[2] Cook, S.: A Hierarchy for Nondeterministic Time Complexity. JCSS, Vol. 4, 1973, No. 7, pp. 343–353.

[3] Geffert, V.: Tally Versions of the Savitch and Immerman-Szelepcsényi Theorems for Sublogarithmic Space. SIAM J. Comput., Vol. 22, 1993, No. 1, pp. 102-113.

[4] Geffert, V.: Sublogarithmic Σ_2-Space Is Not Closed Under Complement and Other Separation Results. Informatique théorique et Applications, Vol. 27, 1993, No. 4, pp. 349–366.

[5] Geffert, V.: A Hierarchy That Does Not Collapse: Alternations in Low Level Space. Informatique théorique et Applications, Vol. 28, 1994, No. 5, pp. 465–512.

[6] Geffert, V.: Bridging Across the $\log(n)$ Space Frontier. Information and Computation 142, 1998, pp. 127–158.

[7] Geffert, V.: Space Hierarchy Theorem Revised, Proc. of MFCS'2001, LNCS 2136, pp. 387–397.

[8] Iwama, K.: ASPACE($O(\log\log(n))$ Is Regular. SIAM J. Comput., 22, pp. 136-46, 1993.

[9] Liśkiewicz, M.—Reischuk, R.: The Complexity World below Logarithmic Space. Proc. the Structure in Complexity Theory, 1994, pp. 64-78.

[10] Liśkiewicz, M.—Reischuk, R.: The Sublogarithmic Alternating Space World. SIAM J. on Computing 25, 1996, pp. 828–869.

[11] Ranjan, D.—Chang, R.—Hartmanis, J.: Space Bounded Computations: Review and New Separation Results. Theoret. Comput. Sci., 80, pp. 289–302, 1991.

[12] Seiferas, J. I.: Relating Refined Space Complexity Classes, JCSS 14, 1977, No. 1, pp. 100–129.

[13] Seiferas, J. I.—Fischer, M. J.—Meyer, A. R.: Separating Nondeterministic Time Complexity Classes, J. of ACM 25, 1978, No. 1, pp. 146–167.

[14] Sudborough, I. H.: Separating Tape Bounded Auxiliary Pushdown Automata Classes. Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, Boulder, Colorado, May 2–4, 1977, pp. 208–217.

[15] Szepietowski, A.: Turing Machines with Sublogarithmic Space. Springer-Verlag, LNCS 843.

[16] Žák, S.: A Turing Machine Space Hierarchy. Kybernetika (Prague) 15, 1979, No. 2, pp. 100–121.

[17] Žák, S.: A Turing Machine Oracle Hierarchy. Proc. MFCS'1979, ed. J. Bečář, LNCS 74, pp. 542–551.

[18] Žák, S.: A Turing Machine Oracle Hierarchy I, II. Comm. Math. Univ. Carol. 21, 1980, pp. 11–39.

[19] Žák, S.: A Turing Machine Time Hierarchy. Theoretical Computer Science, Vol. 26, 1983, pp. 327–333.

Stanislav Žák was born in 1952. He received the Master degree from Charles University Prague, Faculty of Mathematics and Physics, in 1975, and the PhD. degree from the Mathematical Institute of the Academy of Sciences, Prague in 1979. Since 1994 he has been with the Institute of Computer Science of the Academy of Sciences of the Czech Republic, Prague as a research scientist. His research interests are in computational complexity, especially theory of branching programs.