ELSEVIER

Theoretical Computer Science 157 (1996) 267–271

Theoretical Computer Science

Computational complexity of functions[1]

Leonid A. Levin *

Computer Science Department, Boston University, 111 Cummington Street, Boston, MA 02215, USA

Abstract

Below is a translation from my Russian paper. I added references, unavailable to me in Moscow. Similar results have been also given in [9] (see also [6]). Earlier relevant work (classical theorems like Compression, Speed-up, etc.) was done in [15, 13, 2, 1, 14, 7].

I translated only the part with the statement of the results. Instead of the proof part, I appended a later (1979, unpublished) proof sketch of a slightly tighter version. The improvement is based on the results of Meyer and Winklmann [8] and Sipser [12]. Meyer and Winklmann extended earlier versions to machines with a separate input and working tape, thus allowing complexities smaller than the input length (down to its log). Sipser showed the space-bounded Halting Problem to require only additive constant overhead. The proof in the appendix below employs both advances to extend the original proofs to machines with a fixed alphabet and a separate input and working space. The extension has no (even logarithmic) restrictions on complexity and no overhead (beyond an additive constant). The sketch is very brief and a more detailed exposition is expected later [11].

1. Some remarks

We formulate the theorems in terms of the Turing Machine space. But it is clear how to generalize them, since any complexity measure is bounded by a total recursive function (t.r.f.) of any other one. Of course, the accuracy of a constant factor will turn into the accuracy of some other t.r.f. We consider one tape Turing Machines with arbitrary tape alphabets. If the alphabet has n symbols, then input and output integers are written in the n-ary number system. The *space* $p_A(x)$ of an algorithm A is the size (reduced by 1) of the tape used by $A(x)$. The length of a word x is denoted $l(x)$. Obviously, $p_A(x)+1 \geqslant \max(l(x), l(A(x)))$.

The space complexity of any function can be reduced by any constant factor, by extending the alphabet. The inequality within a constant factor $f \prec g$ means $\exists C \forall x$ $f(x) \leqslant Cg(x)$.

[1] Partial translation from [4] (preliminary version is in [3]).

* E-mail: Lnd@bu.edu. Supported by NSF grant CCR-9015276.

SSDI 0304-3975(95)00163-8

Every function F is associated with a class of algorithms that compute it and with the class of their space complexities M_F. We characterize all such classes extending well-known Compression and Speed-up Theorems. Some computable functions do not belong to any class M_F:
Note: A partial function p can be a space of an algorithm if and only if it is itself computable within space $p(x)$. We call such functions *simple*. This requirement is weak since usual functions p are computable in space $l(p(x)) = \log p(x)$.

We call *simple* an algorithm which outputs its own space. We define $p_A(x) = \infty$ when $A(x)$ does not halt and interpret inequalities with simple functions accordingly. Let us agree that an algorithm computes a function F if it does this everywhere in the intersection of its domain and the domain of F.

2. Formulation of the theorems

For any *simple* function G, compression Theorem [13] provides a function F, computable in exactly those spaces p which are simple and $p \succ G$. We generalize this theorem for an arbitrary recursive G.

Theorem 1. *For any t.r.f. G there exists a t.r.f. F, with range $\{0,1\}$ computable in exactly those spaces p which are simple and $p \succ G$.*

Compression Theorem describes a very special case of t.r.f. In [1] t.r.f. were discovered which have no such exact simple lower bounds of complexity. However, the above generalization of the Compression Theorem already describes the general case and can be inverted as follows.

Theorem 2. *For any t.r.f. F there exists a t.r.f. G such that F is computable in exactly those spaces p which are simple and $p \succ G$.*

Thus, the complexity class of any t.r.f. is organized naturally, despite the Speed-up Theorem. The point is that the set of t.r.f. is richer than the set of simple functions. Naturally, the complexity of an arbitrary t.r.f. cannot be always characterized by a simple function, though it is always characterizable by a t.r.f.

Let us describe the properties of the complexity classes for arbitrary t.r.f. A class M is called *canonical* if:

1. all functions of M are simple, and some of them are total;
2. if f, g, h are simple, $f, g \in M$, and $h \succ \min(f, g)$, then $h \in M$; and
3. the class $\overline{M}$ of simple algorithms, computing the functions of M, is of type Σ_2^0, i.e. can be defined as $(p \in \overline{M}) \iff \exists a \forall b\ R(a, b, p)$, where R is recursive.

Theorem 3. *M is the class M_F of all space complexities of some t.r.f. F iff M is canonical.*

This theorem justifies the following conjecture of A.N. Kolmogorov: for any "good" decreasing sequence of functions p_i there exists a function, computable with such and only such space complexities that exceed some of the p_i's. The Compression and Speed-up Theorems are special cases. This conjecture also describes the general case of complexity as it follows from Lemma 1 and Theorem 3. The above results extend to the case of partial functions.

Theorem 4. *Let A be an r.e. set. Theorems 1–3 remain valid if the term "t.r.f." is replaced everywhere by "partial r.f. with domain A", and inequalities like "$a \succ b$" are restricted to $x \in A$.*

3. Proofs

We call *A-canonical* a class M, satisfying conditions 1–3, as adjusted in Theorem 4.

Lemma 1. *For any A-canonical M, a p.r.f. g exists, non-increasing with k and such that $g(k,x)+l(k)$ is simple, domain of $g(0,x)$ is A and $p \in \overline{M} \iff (\exists k\, g(k,x) \prec p(x))$, for any simple p.*

Appendix (not part of the translation)

Below is the sketch of a proof of a slightly tighter statement. It assumes separate input and working space, thus allowing spaces $o(|x|)$, as in [8]. It also assumes a fixed tape alphabet, allowing additive (rather than multiplicative) constant accuracy. The latter uses the result of [12] that the space s bounded halting problem can be solved in space $s + O(1)$. Otherwise, the version is similar to the above translation. For log of time of a Pointer Machine or of some Turing Machine versions [5] similar results hold.

Model. To allow space limits below the input bit-length $|x|$, one needs to differentiate the input symbols from symbols used as memory during the computation. Instead of separating the input tape as in [2], I prefer to separate the "ink". While not essential, this preserves the simple space-time geometry of the one-tape Turing Machine (TM). So, we separate the state of each cell into a read-only *ink* and a read-write *pencil* part. The ink part cannot be modified after the input is written and is ignored for measuring space. The ink (but not necessarily pencil) string starts at the left end of the tape after exactly one blank. The ink and pencil string and their union form each a continuous segment without blanks. The alphabet is fixed and has at least two symbols $\{0,1\}$ besides the blank. *Space*: $S_{A(x)}$ or $S_A(x)$ is the supremum of bit-lengths of the pencil string throughout the computation of $A(x)$. The output may either be left on the tape/head or its digits "flashed" sequentially at the (fixed) leftmost cell. In some cases the pencil string starts not empty. E.g. g-constructible functions f are those computable in space $\max\{t, f(x)\}$ starting from input x and any pencil string of length $t \geqslant g(x)$; for $g = 0$ we omit the prefix "g-" and for $g = f$ replace it with "semi-".

Conventions. Let $U(k,x)$ be a universal TM with $\{0,1\}$ outputs. It ignores the "padding" k_2 in its program $k = (k_1, k_2)$. Appropriate paddings can put any Σ_2^0 program set in the form $m = a^{-1}(\{\infty\})$ for some function a with constructible $a(k) - 4|k|$. Let $p(k,x) \stackrel{\text{def}}{=} 4|k| + S_U(k,x)$ and expressions like $p_k(x)$ mean $p(k,x)$. Assume $0 \in m$ and $M = \{p_k : k \in m\}$. Consider the closure $\overline{M}$ of a set M of functions under inclusion of each h s.t. for some $f, g \in M$, $h \geqslant \min\{f,g\} - 1$ in the domain D of $U(0,x)$. Call sets M_1, M_2 *confinal* if their closures contain the same constructible functions. Define $[a < b]$ as a, if $a < b$ and 0 otherwise; likewise for $\leqslant$. Clearly, the complexity class of any function can be described as $\overline{M}$ above.

Construction. Now we build (cf. Lemma 1) a monotone sequence g_k confinal to M: If $a(k) > t > p(k,x)$, let $p^t(k,x) \stackrel{\text{def}}{=} \max_{l<k}\{p(k,x), [a(l) \leqslant t]\} \leqslant t$. Otherwise, $p^t(k,x) \stackrel{\text{def}}{=} t$. Then, $g(1,x) \stackrel{\text{def}}{=} p(0,x)$; $g(k+1,x) \stackrel{\text{def}}{=} p^{g(k,x)}(k,x)$; $g_\infty(x) \stackrel{\text{def}}{=} \min_k g(k,x)$; $k_x \stackrel{\text{def}}{=} \min\{l : g(l,x) = g(k,x)\}$. To compute $g_k(x)$ we carry k, $g_{k-1}(x)$ as the pencil string length, and the largest relevant $a(l)$ as $g_{k-1}(x) - a(l)$ (if $< 2|k|$) or as l. Cutting the values of p, a to the maximum of t would not affect those values of g below t. So, $g(k,x) - 2|k_x|$ is g_∞-constructible; g_k are uniformly recursive with domain D and equal $\min_{l<k}\{p_l(x) : l \in m\}$, on D, except when both are $\leqslant \max_{l<k}\{[a(l) < \infty]\} = \mathrm{O}(1)$.

Next we convert such $\{g_k\}$ into a confinal set consisting of a single semiconstructible recursive function G on D (cf. Theorem 2): Let $b(k) \stackrel{\text{def}}{=} \min_x(2|k,x| + g_1(x) : g_k(x) > p_k(x))$ and $K(x) \stackrel{\text{def}}{=} \min\{k : b(k) > g_k(x) > p_k(x)\}$. Then $G(x) \stackrel{\text{def}}{=} g(K(x),x) \leqslant \max_{l \leqslant k}\{g(k,x), [b(l) < \infty]\}$, for all k.

Conversely, $G < p_k$ in D implies $b(k) = \infty$. Indeed, $b(k) = 2|k,x| + g_1(x) < \infty$ while $g_k(x) > p_k(x)$ yields $K(x) \leqslant k$ and $G(x) \geqslant g_k(x) > p_k(x)$. $b(k) = \infty$ makes $g_k(x) \leqslant p_k(x)$ in D and $p_k \in \overline{M}$.

Finally, for such G, we build a G-constructible predicate $\Pi(x) \stackrel{\text{def}}{=} 1 - U(K'(x),x)$ with complexity class confinal to G (cf. Theorem 1): Here $K'(x) \stackrel{\text{def}}{=} \min\{k : c(k) > G(x) \geqslant p_k(x)\}$ and $c(k) \stackrel{\text{def}}{=} \min_x\{2|k,x| + p_0(x) : K'(x) = k\}$.[1] If $\Pi(x) = U(k,x)$ in D then $k \notin K'(D)$ and $c(k) = \infty$. Then $G(x) \leqslant \max_{l<k}\{p_k(x), [c(l) < \infty]\}$ and $p_k \in \overline{M}$.

Acknowledgements

My work on this extension, as well as my smooth implantation into the Western scientific community was made possible by comprehensive support and encouragement by Albert Meyer in 1978–80. I am among the many greatly indebted to his concern and generosity.

[1] A leaner version: $c(k) \stackrel{\text{def}}{=} \min_x\{2|k,x| + \max\{p_0(x), p_k(x)\} : \Pi(x) \neq U(k,x)\}$.

References

[1] M. Blum, A machine-independent theory of the complexity of recursive functions, *J. ACM* **14** (1967) 322–336.

[2] J. Hartmanis and R.E. Stearns, On the computational complexity of algorithms, *Trans. AMS* **117** (1965) 285–306.

[3] L. Levin, On storage capacity for algorithms, *DAN SSSR = Soviet Math. Dokl.* **14** (1973).

[4] L. Levin, Computational complexity of functions, in: V.A. Kosmidiadi, N.A. Maslov and N.V. Petri. eds., *Complexity of Algorithms and Computations* (Mir, Moscow, 1974) 174–185.

[5] L. Levin, Theory of computation: how to start, *SIGACT News* **22** (1991) 47-56.

[6] N. Lynch, "Helping": several formalizations, *J. Symbolic Logic* **40** (1975) 555–566.

[7] A.R. Meyer and P.C. Fischer, Computational speed-up by effective operators, *J. Symbolic Logic* **37** (1972) 55–68.

[8] A.R. Meyer and K. Winklmann, The fundamental theorem of complexity theory (preliminary version), in: J.W. de Bakker and J. van Leeuwen, eds., *Foundations of Computer Science III, Part 1: Automata, Data Structures, Complexity*, Mathematical Centre Tracts 108 (Amsterdam, 1979) 97–112. Also a draft, 1978.

[9] C.P. Schnorr and G. Stumpf, A characterization of complexity sequences, *Z. Math. Logik Grundlagen Math.* **21** (1975) 47–56.

[10] J.I. Seiferas, Machine-independent complexity, In: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity* (Elsevier, The MIT Press, Amsterdam, 1990) 163–186.

[11] J.I. Seiferas and A.R. Meyer, Characterization of reliable space complexities, in preparation.

[12] M. Sipser, Halting space-bounded computations, *Theoret. Comput. Sci.* **10** (1980) 335–338. Also, FOCS-1978.

[13] M. Rabin, Speed of computation and classification of recursive sets, *Third Conv. Scient. Societies* 1–2, Israel, 1959 (also Tech. Report. 3, Hebrew Univ., Jerusalem, 1960).

[14] B.A. Trakhtenbrot, Complexity of algorithms and computations, Course notes in Russian (Novosibirsk University, USSR, 1967).

[15] G.S. Tseitin, Talk on Math. Logic Seminar, Moscow University, 1956. Also pp. 44–45 in: S.A. Yanovskaya, Math. Logic and Foundations of Math., *Math. in the USSR for 40 Years*, 1:13–120 (Moscow, Fizmatgiz, 1959) (in Russian).