1400008561
còpia 1

Generalized Kolmogorov complexity in relativized separations

Ricard Gavaldá
Leen Torenvliet
Osamu Watanabe
José L. Balcázar

Report LSI–90–21

FACULTAT D'INFORMÀTICA
BIBLIOTECA
R. 7912 07 JUN. 1990

Abstract. We describe several developments of a technique, due to Hartmanis, that uses Kolmogorov complexity to prove the existence of relativizations separating complexity classes. The main advantage of these proofs is that they clearly show the limitations of certain classes of oracle machines and the relevance of these limitations for the proof. Such limitations refer to the extent to which the machines defining the class are able to process Kolmogorov-complex structures.

Resum. Presentem diverses extensions d'una tècnica introduïda per Hartmanis que fa servir complexitat de Kolmogorov per a demostrar l'existència de relativitzacions que separen classes de complexitat. El principal avantatge d'aquestes tècniques és que mostren clarament les limitacions d'algunes classes de màquines amb oracle, i la importància d'aquestes limitacions per a la demostració. Aquestes limitacions afecten la capacitat de les màquines que defineixen la classe per a tractar estructures complexes en el sentit de Kolmogorov.

Resumen. Presentamos varias extensiones de una técnica introducida por Hartmanis que utiliza complejidad de Kolmogorov para demostrar la existencia de relativizaciones que separan clases de complejidad. La principal ventaja de estas técnicas es que muestran claramente las limitaciones de algunas clases de máquinas con oráculo, y la importancia de estas limitaciones en la demostración. Estas limitaciones se refieren a la capacidad de las máquinas que definen la clase para tratar estructuras complejas en el sentido de Kolmogorov.

GENERALIZED KOLMOGOROV COMPLEXITY IN RELATIVIZED SEPARATIONS †

(Preliminary version, April 1990)

Ricard Gavaldà *, Leen Torenvliet **, Osamu Watanabe ***, and José L. Balcázar *

Abstract. We describe several developments of a technique, due to Hartmanis, that uses Kolmogorov complexity to prove the existence of relativizations separating complexity classes. The main advantage of these proofs is that they clearly show the limitations of certain classes of oracle machines and the relevance of these limitations for the proof. Such limitations refer to the extent to which the machines defining the class are able to process Kolmogorov-complex structures.

1. Introduction

We describe several applications of generalized Kolmogorov complexity to show separations of relativized complexity classes. Our emphasis is not so much on the results, which can be obtained by more usual techniques like slow diagonalizations, but in the clear explanation provided by the techniques used here: each proof makes apparent a weakness in the computational power of a class of oracle machines.

We follow Hartmanis' [8] definition of resource-bounded Kolmogorov complexity. The use of this concept to prove relativized separation results is very advantageous: the

† The work of R. Gavaldà and J.L. Balcázar was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

Authors' adresses:

* Departament de Llenguatges i Sistemes Informàtics, Univ. Politècnica de Catalunya. 08028 Barcelona, Spain.

** Department of Mathematics and Computer Science, Universiteit van Amsterdam. Plantage Muidergracht 24, 1018 TV Amsterdam.

*** Department of Computer Science, Tokyo Institute of Technology. Meguro-ku, Ookayama, Tokyo, Japan 152.

power of a class of oracle machines can be clearly described by the extent to which these machines are able to create and process random structures. This idea is already used in Hartmanis' paper, in which a relativized separation P $\neq$ NP is described.

For completeness, we begin by describing the use made by Hartmanis of this technique, providing another example. Then we present some further developments of this technique. First, we study oracles that give information on how to create complex queries. This allows us to distinguish between adaptive and nonadaptive machines, showing certain limitations of nonadaptive oracle access. We also consider the ability to record all queries and oracle answers on work tapes. This will show certain limitations of space-bounded machines. Finally, we indicate how the oracle can contain additional information that can be retrieved only if a complex string (the "password") is previously found, and how this idea combines with the abilities just enumerated.

For each technique, we give an intuitive explanation and some examples of application, all related to the space-bounded classes PSPACE and NPSPACE and the time-bounded class EXPTIME. Relativized comparisons of these classes may be obtained by more traditional techniques like ad-hoc diagonalizations, although to our knowledge not many of such constructions have been published. A forthcoming publication [7] will show how to combine these techniques with other structural properties such as printability to show that all the noncontradictory relationships between PSPACE, NPSPACE, and EXPTIME hold in the adequate relativization. Results using similar techniques can be found in [5] and [14]. Generalized Kolmogorov complexity is also used in [1] and [2] to achieve separations between relativized complexity classes.

2. Preliminaries

Our computational model is the multitape oracle Turing machine with distinguished QUERY, YES, and NO states. We will consider only time and space bounded machines. In the case of space bounds, the oracle tape is not considered to be bounded. This unboundedness of the oracle tape is important since otherwise most of our results are easily seen to hold for simple reasons; even stronger results for such a model appear in [12]. In deterministic machines the oracle tape is bounded a fortiori by the time bound, which is at most exponential on the space bound. In the nondeterministic case, we explicitly rule out unbounded computations and force the same exponential bound.

Define EXPTIME as the class of languages that can be recognized in deterministic $2^{n^{O(1)}} + O(1)$ time. Definitions of PSPACE and NPSPACE are standard. For undefined concepts and notations see [3]. Related background notions will appear in [4].

We assume an encoding of instantaneous descriptions (i.d.'s) of machines such that the length of the input to an i.d. is always visible. The oracle tape of a space bounded machine is not included in its i.d.'s. The i.d.'s with state QUERY are called *query i.d.*'s. Let M be an oracle machine, c_1 any i.d. and c_2 a query i.d. of M. For a word u we say that (c_1, c_2) *generates* u if M can go from i.d. c_1 to i.d. c_2 writing u on the oracle tape

without going through the QUERY state (except in c_2).

We say that a set is *spaced out* by a function between natural numbers $s(n)$ if it is empty at lengths that are not of the form $s(n)$ for any n.

For a natural n, let $bin(n)$ be the binary representation of n. For a string $w \in \{0,1\}^*$, $\overline{w}$ is the string that results from doubling each bit of w. The string $w' = \overline{bin(|w|)}01w$ is called the *self-delimiting* version of w. We will denote with $x\#y$ the string $x'y'$, and use $\#$ as a pairing function. It can be seen that $|x\#y| = |x| + |y| + O(\log |xy|)$.

Generalized or resource bounded Kolmogorov complexity was introduced by several authors in slightly different ways; see [11] for a complete survey. We will follow the definition of Hartmanis [8]; some results related to this concept will be presented also in [4]. We will use the space bounded version. Fix some universal machine U. For any two natural numbers n, m, a string x is in the (finite) set $KS[n,m]$ if U with some description of length at most n as input and running in space m gives x as output. We will say that a string of length n is *random* if it is not in $KS[n-1,m]$ for some m that will be clear from the context.

Let $w(n)$ denote the first string of length n in the lexicographical order that is not in $KS[n-1,2^n]$. The 2^n space bound is not critical: it is only used to ensure that words $w(n)$ can be found recursively if necessary. In fact, it can be seen that by exhaustive search, $w(n)$ can be found from input n in space $O(2^n)$.

3. The ability to create complex queries

In this section we discuss the technique already used in [8] to distinguish between determinism and nondeterminism, giving a different example of application. The key idea is that nondeterministic machines can construct complex queries by exploiting nondeterminism, but a deterministic machine cannot generate complex queries by itself: each query is generated in a deterministic computation and can be uniquely reconstructed from the i.d. of the machine when it starts to compute it. This is the weakness we will use.

We present an oracle for which NPSPACE $\not\subset$ EXPTIME; this also means that PSPACE $\neq$ NPSPACE for this particular oracle, and is another example of a known fact: Savitch's theorem does not relativize with unbounded oracle tape.

To construct the oracle, take some tally set that is too difficult for EXPTIME machines, but easy enough to be decided if more resources are allowed. The oracle will tell whether a certain word is in the tally set only if we know a word of related length that indicates it. If this word is complex, and cannot be constructed from the answers to other queries, a deterministic machine will not be able to manufacture it. It will then be as helpless as a non-oracle machine. To ensure that other, smaller, queries do not give much information, we will keep words in the oracle largely separated.

1. Theorem. There is an oracle A such that NPSPACE(A) $\not\subset$ EXPTIME(A).

Proof. Construct by diagonalization a tally set $T \notin$ EXPTIME, but decidable within some larger time bound, for instance double exponential time. At the same time, make T spaced out by some function $s(.)$ much larger than a double exponential, i.e., T only contains words of the form $0^{s(m)}$. If $s(.)$ grows fast enough, a machine on input $0^{s(m)}$ can perform the construction of T up to length $s(m-1)$ in $o(\log(s(m)))$ space. Define $A = \{w(2^n) \mid 0^n \in T\}$.

Notice first that $T \in$ NPSPACE(A) by the following algorithm: On input 0^n, guess a word of length 2^n and accept iff it is in the oracle.

We claim that $T \notin$ EXPTIME(A): Assume that an EXPTIME machine E running in time 2^{n^i} decides T with oracle A. We will give a deterministic machine for T that skips all oracle queries.

On input 0^n, with $n = s(m)$, simulate $E(0^n)$ solving oracle queries as follows: for a query u of length 2^l, with $l \leq s(m-1)$, test that $0^l \in T$ (with a slow algorithm for T) and that $u = w(2^l)$ (using a slow algorithm to generate $w(2^l)$). For longer queries, assume a NO answer.

It is clear that this procedure answers "small" queries correctly. For other queries, it could only be mistaken if it queried precisely for $w(2^n)$, which only has descriptions of size at least 2^n. We show that all queries of length 2^n have much smaller complexity, so the algorithm correctly assumes a NO answer. Let t be a table of A up to length $s(m-1)$. Then the l-th query of $E(0^n)$ can be reconstructed from the descriptions of n, t, and l: Run the computation of $E(0^n)$ up to the l-th query using t to answer the other queries. Since t has size smaller than n and E can make at most 2^{n^i} queries, $|n\#t\#l| < n + n + n^i < 2^n$ (if n is somewhat large).

This procedure does not query any oracle and, if $s(.)$ grows fast enough, works in exponential time. This means that $T \in$ EXPTIME, which is in contradiction to the way T was chosen. ∎

4. The ability to create complex queries with oracle guidance

The technique discussed in this section allows us to separate relativized complexity classes using adaptiveness properties. There is no formal definition of the concept of "adaptiveness". Intuitively, an adaptive machine is able to ask queries that inherently depend on the answers to previous queries. For instance, polynomial time truth-table reducibility [10] is nonadaptive by definition, since it corresponds to querying the oracle after deciding all the queries to be made (such a behavior has been called also "parallel queries"). On the other hand, polynomial time Turing reducibility is adaptive, since it can ask queries that depend of previous answers, and this fact allows one to prove that it differs from polynomial time truth-table reducibility [10]. On the contrary, nondeterministic polynomial time Turing reducibility is nonadaptive, since it coincides with nondeterministic

polynomial time truth-table reducibility [10]. The reason is that nondeterminism allows the machine to figure out the answers in advance, and to compute the queries using the guessed answers.

The fact that space-bounded machines suffer from a lack of adaptiveness (or at least from a limitation on their adaptiveness) due to the impossibility of recording many answers has been observed before; e.g. Wagner [13] shows that logarithmic space Turing reducibility to NP is the same as polynomial time truth-table reducibility to NP. Discussions on the adaptiveness of logspace machines and its relationship to circuit classes and nonuniformity can be found in [6]. Here we use generalized Kolmogorov complexity to make apparent the weakness that arises from lack of adaptiveness.

The general idea of these results is the following: suppose that acceptance depends on a particular word that is hard to construct. If the oracle gives enough information on how to find this word, and a machine can record all the answers, it will be able to find it. In other words, a machine can construct long random structures because it can "extract complexity" from the oracle piece by piece. But a space bounded machine is weaker because it cannot build the complex word to be queried.

The example we present here is a separation of EXPTIME from PSPACE. This result also follows from a construction in [9]. They give a tally set in P – DLOG (relativized), so the set of indexes of that tally set is in EXPTIME – PSPACE (relativized). In our construction, the set in EXPTIME – PSPACE is itself a tally set.

As in the last section, we will take a tally set that is too difficult for PSPACE and encode it with some complex words in the oracle. Now, the oracle will also contain all the prefixes of these words. An EXPTIME machine can find the word because it can follow all the prefixes, but a PSPACE machine, that can only remember a small amount of information between queries, cannot build complex queries using little space. The idea of including in the oracle the prefixes of complex words is used in [14] to show that polynomial time Turing and truth-table reducibilities differ in EXPTIME.

2. Theorem. There is an oracle B such that $\mathrm{PSPACE}(B) \subsetneq \mathrm{EXPTIME}(B)$.

Proof. Define the set $PREF = \{0^n\#v \mid v \text{ is a prefix of } w(n)\}$. Since words $w(n)$ can be found in $O(2^n)$ space, $PREF$ can be decided in $O(2^n)$ space by exhaustive search. Construct by diagonalization a tally set $T \notin \mathrm{EXPTIME}(PREF)$, that should be decidable, for instance, within quadruple exponential time by reconstructing $PREF$. Also make T spaced out more than a quadruple exponential. Define $B = PREF \oplus \{w(2^n) \mid 0^n \in T\}$.

With the help of $PREF$, $T \in \mathrm{EXPTIME}(B)$: On input 0^n, expand bitwise $0^{2^n}\#$ to find $w(2^n)$ and accept iff it is in the oracle.

However, T is not in $\mathrm{PSPACE}(B)$. To show this, assume that a PSPACE machine M that works in n^i space decides T with oracle B. On input 0^n, every query that M constructs is uniquely determined by the contents of its work tapes after the previous oracle query (or at the beginning of the computation). Since it uses no more than n^i space and no

description of $w(2^n)$ can be so short, M cannot construct and query $w(2^n)$.

Now it is possible to give a PSPACE algorithm that only queries $PREF$ and decides T, giving a contradiction. This is done like in the previous theorem. ■

5. The ability to create complex queries and know it

We have seen in section 3 that nondeterministic machines can generate complex queries. Take for instance the set not in PSPACE that we constructed in the last section. This set is in fact in NPSPACE, because a nondeterministic machine can find the coding word without following the prefixes. Yet, the exponential time machine in the construction has an advantage over any nondeterministic polynomial space machine: it can keep the complex query stored in worktape and thereby perform a deterministic construction, so that the machine knows that the construction succeeded. The nondeterministic machine can find it nondeterministically, but cannot distinguish it from other queries, and cannot be sure of whether the construction succeeded.

In this section we exploit this limitation of nondeterministic machines: when they try to generate a complex query they may succeed, but they cannot know whether they succeed. More precisely, they can guess and ask a very long and difficult query, but only at the price of confusing it with many more different queries. We prove this fact in a technical lemma.

3. Lemma. Let N be a NPSPACE machine that works in space n^i, and let c_1, c_2 be i.d.'s of N over the same input of length n. Suppose (c_1, c_2) generates some word u of length 2^n not in $KS[2^n - 1, 2^{2^n}]$. Then (c_1, c_2) generates at least $2^{2^n - 3n^i}$ words of length 2^n.

Proof. Suppose that there are k such words and that u is the l-th of them in lexicographical order ($l \leq k$). Then, from description $c_1 \# c_2 \# l$, we can recover u: For every word of length 2^n, explore the whole computation tree of N between c_1 and c_2, and find whether the word is generated; stop when the l-th word that satisfies this condition is found. All this can be done in space 2^{2^n}. Since, by the hypothesis on u it must be that $2^n \leq |c_1 \# c_2 \# l| \leq |c_1 \# c_2 \# k| \leq 3n^i + |k|$, we have $k \geq 2^{2^n - 3n^i}$. ■

Now the main point is the following: using oracle guidance, an exponential time machine can find the long complex string that must be asked, and it knows that the string is the correct one. On the other hand, a nondeterministic machine can guess the complex string and trust a YES answer, but cannot trust a NO answer since it may have failed to guess the right string. So if one bit is encoded in the oracle as the absence of the complex string, instead of encoding it as its presence, this bit is no longer accessible to nondeterministic space-bounded machines.

4. Theorem. There is an oracle C such that EXPTIME(C) $\not\subseteq$ NPSPACE(C).

Proof. Define the set $PREF$ as in the last section, and construct by diagonalization a tally set T that is not in NPSPACE($PREF$). Again, make sure that T can be decided within some large time bound and that it is spaced out by a larger function $s(.)$, so that small words in T can be found easily enough. The oracle C is defined as $PREF \oplus \{w(2^n) \mid 0^n \notin T\}$.

The set T is in EXPTIME(C), because an EXPTIME machine can find $w(2^n)$, query it and then accept iff it receives a NO answer.

We show now that $T \notin$ NPSPACE(C). Suppose to the contrary that T is accepted by machine N with oracle C in space n^i. We claim that N would not really need $w(2^n)$ in the oracle to decide 0^n. Call $w = w(2^n)$ for simplicity.

Claim: For n large, $N(0^n)$ accepts with oracle $C \cup \{1w\}$ iff it accepts with oracle $C - \{1w\}$.

Proof of the Claim. From left to right, since N^C accepts T, if $1w \in C$, N cannot accept 0^n with oracle $C = C \cup \{1w\}$. If $1w \notin C$, it is true that N accepts 0^m with oracle $C = C - \{1w\}$.

From right to left: Suppose $N(0^n)$ accepts with oracle $C - \{1w\}$. If there is an accepting computation path that does not query $1w$, this path still accepts with oracle $C \cup \{1w\}$. Otherwise, choose an accepting path and let (c_1, c_2) be the pair of i.d.'s that generates w the first time that $1w$ is queried on this path. Since w has length 2^n and is random, by lemma 3, there are many other words v that are also generated from (c_1, c_2).

But N gets a NO answer both from $1w$ and $1v$ (because none of them is in $C - \{1w\}$), so after these queries N is exactly in the same configuration. This means that the path that generates $1v$ is also accepting. Repeating this argument for all queries to $1w$ along the path, we get an accepting path that does not query $1w$. This path will accept even if $1w$ is added later to the oracle. ■

Consider now the following NPSPACE machine N', that behaves like N^C but skips all queries to the oracle:

On input 0^n, for n large, test first that $n = s(m)$. Simulate N until it queries its oracle on a word $1u$. If $s(m-1) \geq |u|$, then u is so small that $1u \in C$ can be decided directly by the algorithm for T in space polynomial in n. Otherwise, $1u$ could be in C only if $u = w(2^n)$. Assume a NO answer without even asking the query to the oracle.

N' with oracle $PREF$ thus simulates N with oracle $C - \{1w\}$, which by the claim is equivalent to using oracle C. We have a NPSPACE machine that accepts T with oracle $PREF$, which is in contradiction to the way T was constructed. ■

Note that the set T is however in coNPSPACE(C), and so NPSPACE(C) is not closed under complements. This is also true if the set of prefixes is not added to the oracle. So, NPSPACE $\neq$ coNPSPACE in a relativized world does not imply in itself that EXPTIME $\not\subseteq$ NPSPACE in that world, because in the given example the set $T \in$ coNPSPACE is not in EXPTIME without the guiding prefixes.

6. Complex strings as "passwords"

The preceding constructions can be understood as encoding a tally set into the oracle, protecting each bit of the characteristic function of the tally set by a complex "password" that only certain machines can find. Similarly, other information can be encoded in the oracle protected by a complex prefix. As an example, we use this fact to give an oracle that makes PSPACE $\subsetneq$ NPSPACE = EXPTIME.

In this case, we will add an EXPTIME-complete set to a set that makes PSPACE $\subsetneq$ EXPTIME. Every word of the complete set will be prefixed by a complex word. The words in the complete set will then be "hidden" from all PSPACE machines. Since they will not be able to access the set (except on small lengths), we will keep PSPACE $\subsetneq$ EXPTIME. On the other hand, NPSPACE machines can guess any word they need to solve any EXPTIME predicate.

5. Theorem. There is a set D such that PSPACE(D) $\subsetneq$ NPSPACE(D) = EXPTIME(D).

Proof. Take the set B in section 4 such that PSPACE(B) $\subsetneq$ EXPTIME(B). Recall that B was the join of $PREF$ with another set, and that there was a tally set $T \in$ EXPTIME(B) − PSPACE(B).

To construct a complete set, fix an effective enumeration of clocked EXPTIME machines, $\{E_i\}$, such that any E_i runs in time at most 2^{n^i} on inputs of size n. Define D as $B \oplus \{w(m)\#i\#x \mid m = 2^{|x|^i}$ and E_i^B accepts x (in time $m)\}$.

We show first that, still, $T \notin$ PSPACE(D). Suppose, to the contrary, that a PSPACE machine M accepts T with D as an oracle, and runs in space n^i. Consider the simulation of this machine on input 0^n, with oracle B instead of D and solving queries to the second part of D as follows:

- Queries of the form $u\#j\#y$ to D, with $|u| \leq n^i$. Then it is possible to simulate $E_j^B(y)$ in $O(n^i)$ additional time (and space). With the help of $PREF$, test also that $u = w(2^{|y|^j})$.

- Queries of the form $u\#j\#y$ to D, with $|u| > n^i$. A NO answer can be assumed since u cannot be $w(2^{|y|^j})$: it has been generated from an i.d. of M of size $n^i < |u|$.

So, T can be decided by a PSPACE machine that only uses B as an oracle. This is not possible by the assumptions about T and B. Since $T \in$ EXPTIME(B) and B is one of the components of D, we have shown that PSPACE(D) $\neq$ EXPTIME(D).

Now we prove that NPSPACE(D) can decide any EXPTIME(D) predicate. First note that EXPTIME(D) = EXPTIME(B): If a machine has time enough to query $w(2^{|x|^i})\#i\#x$ to oracle D, the query can be solved in time $O(2^{|x|^i})$ with oracle B, and so writing a query down takes roughly as much time as solving it. Now it is clear that EXPTIME(B) $\subseteq$ NPSPACE(D): To solve whether $E_i(x)$ accepts with oracle B, a nondeterministic machine guesses the word $w(2^{|x|^i})$ on the oracle tape, writes $\#i\#x$ and finally accepts iff it gets a YES answer from D.

It remains to show that NPSPACE(D) $\subseteq$ EXPTIME(D). Consider any nondeterministic machine N that runs in space n^i on inputs of size n, and so can query words of length up to 2^{n^i}. We claim that there is an EXPTIME(D) machine that, on input 0^n, builds a sorted table of D up to length 2^{n^i}.

This is easy for B since it is closed under prefixes. The other component of D contains words of the form $w(2^m)\#j\#y$, with $|w(2^m)| = 2^m$ and $|j\#y| \leq m$. So, for any m there are at most 2^m words of the form $1w(2^m)\#j\#y$ in D.

The claimed machine, on input n and for all $m \leq n^i$: 1) Finds $w(2^m)$ using $PREF$, 2) scans all words $1w(2^m)\#j\#y$ and adds to the table those that are in D and have length $\leq 2^{n^i}$, and 3) sorts the resulting table. This procedure captures all D and some routine calculations show that it runs in time polynomial in 2^{n^i}.

Define now the set

$$L_N = \{x\#l\#0^m \mid m = 2^{|x|^i} \text{ and } N \text{ accepts } x \text{ using the sorted list } l \text{ as an oracle } \}$$

With a little care, words in L_N can be decided in nondeterministic logspace: On input $x\#l\#0^m$, test first that l is a sorted list $l_1\#l_2\#\ldots\#l_k$ (this can be done in logspace) and simulate $N(x)$; whenever N starts writing on its oracle tape, guess nondeterministically an index $u < k$. We will test that the query word that N is going to write is lexicographically between l_u and l_{u+1}.

When N writes a new bit on the oracle tape, test that the query that N is writing can still be between l_u and l_{u+1}. If this is not true, abort the computation and reject (u was a wrong guess). When N finally enters its QUERY state, assume a YES answer if it has exactly written either l_u or l_{u+1}. Otherwise, since the query is strictly between l_u and l_{u+1}, it is not in l and a NO answer can be assumed.

All these tests can be done locally and there is no need to keep the whole query, so this simulation uses the same space as N plus some pointers to the list l. This is about $|x|^i + O(\log |l|) = O(\log |x\#l\#0^m|)$, which means $L_N \in$ NLOG $\subseteq$ P. Then, for every N, the following EXPTIME(D) machine accepts exactly the same language as N^D: On input x, obtain the accessible part of oracle D, l, and run the P algorithm for L_N on $x\#l\#0^m$, with $m = 2^{|x|^i}$. ∎

See [7] for a more general discussion of the oracles such that NPSPACE $\subseteq$ EXPTIME, as well as some conditions that make Savitch's theorem work.

7. Acknowledgement

The interchange of ideas that finally crystallized in this paper began at a workshop organized by Ron Book at UCSB in june 1987. Thanks are due to him for providing two of the authors with periodic opportunities to discuss about research.

8. References

[1] E. Allender: "Limitations of the upward separation technique". In: *Proc. 16th Int. Coll. Automata, Languages, and Programming* (1989), 18–30. Springer-Verlag Lecture Notes in Computer Science 372.

[2] E. Allender and C. Wilson: "Downward translations of equality". Rutgers University Technical Report DCS–TR–258, 1989.

[3] J.L. Balcázar, J. Díaz, and J. Gabarró: *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science, vol. 11 (1988), Springer-Verlag.

[4] J.L. Balcázar, J. Díaz, and J. Gabarró: *Structural Complexity II.* EATCS Monographs on Theoretical Computer Science, vol. 22 (1990), Springer-Verlag (in press).

[5] R. Book, P. Orponen, D. Russo, and O. Watanabe: "On exponential lowness". *SIAM Journal on Computing* **17** (1988), 504–520.

[6] H. Buhrman, E. Spaan, and L. Torenvliet: "On adaptive resource bounded computations" (submitted for publication).

[7] R. Gavaldà: "Separations of exponential time and polynomial space". Research report, LSI Department, Univ. Politècnica de Catalunya (in preparation).

[8] J. Hartmanis: "Generalized Kolmogorov complexity and the structure of feasible computations". In: *Proc. 24th IEEE Symposium on Foundations of Computer Science* (1983), 439–445.

[9] R. Ladner and N. Lynch: "Relativization of questions about log space computability". *Mathematical Systems Theory* **10** (1976), 19–32.

[10] R. Ladner, N. Lynch, and A. Selman: "A comparison of polynomial time reducibilities". *Theoretical Computer Science* **1** (1975), 103–123.

[11] M. Li and P.M.B. Vitányi: "Two decades of applied Kolmogorov complexity". In: *Proc. 3rd Structure in Complexity Theory Conference* (1988), 80–101.

[12] P. Orponen: "Complexity classes of alternating machines with oracles". In: *Proc. 10th Int. Coll. Automata, Languages, and Programming* (1983), 573–584. Springer-Verlag Lecture Notes in Computer Science 154.

[13] K. Wagner: "Bounded query computations". In: *Proc. 3rd Structure in Complexity Theory Conference* (1988), 260–277.

[14] O. Watanabe: "A comparison of polynomial time completeness notions". *Theoretical Computer Science* **54** (1987), 249–265.