A positive relativization of polynomial time versus polylog space

Ricard Gavaldà *

Department of Software (L.S.I.)

Universitat Politècnica de Catalunya

08028 Barcelona, Spain

gavalda@lsi.upc.es

Abstract. Can every problem in P be solved in polylogarithmic space? We show that this question is equivalent to whether the class E is included in the class PSPACE relative to any oracle, for a restricted type of oracle machine. We give a similar characterization for the problem of whether polylog space is included in P. The proofs are based on translation arguments different from those used in the literature.

Keywords: Computational complexity, complexity classes, relativization

1. Introduction

This note is concerned with the precise relation between the class of sets decidable in polynomial time (P) and the class of sets solvable in polylog space (PLOG). It is known that these two classes are different: Book [4] observed that P has complete sets for logspace reducibility while PLOG does not. Still, whether either class is included in the other remains an open question.

Our main result is in spirit a *positive relativization* of the "P versus PLOG" question. In most positive relativizations described up to now, however, a question of the type "$\mathcal{C} \subset \mathcal{D}$?" is translated to a question about relativized $\mathcal{C}$ and relativized $\mathcal{D}$. See [2] for a more formal statement. By contrast, we translate the questions about P and PLOG to the relativization of two exponentially higher classes.

The two relativized classes that we compare are PSPACE $= \bigcup_{k>0} \mathrm{DSPACE}(n^k)$ and E $= \bigcup_{k>0} \mathrm{DTIME}(2^{kn})$, defined using the oracle Turing machine as our computation

* This research was partially supported by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II), and by DAAD and MEC through Acciones Integradas 1992, 131-B, 313-AI-e-es/zk. The paper was prepared while visiting the University of California at Santa Barbara, supported in part by the National Science Foundation under grant CCR89-13584.

model. To relativize the class PSPACE, it is important to define precisely the way of accessing the oracle, because different types of access give classes with very different properties. Hartmanis [7] discusses several ways of relativizing space classes and, in particular, the anomalous behavior of some of the models.

Here, we impose a linear bound on the length of the oracle tape: we consider space-bounded machines that, on any input of length n, query strings of length at most cn to their oracles. Furthermore, the constant c does not depend on the oracle, but on the machine only. Let $\mathrm{PSPACE}_{lin}(A)$ be the class of sets accepted by PSPACE machines with this restriction, relative to oracle A.

It can be shown that the anomalies discussed in [7] do not appear with our definition of PSPACE_{lin}. For example, every set A is clearly in $\mathrm{PSPACE}_{lin}(A)$, which is false if only queries of sublinear length are allowed. Also, as for unrelativized computation, adding nondeterminism to PSPACE_{lin} machines does not increase their computational power. If we define $\mathrm{NPSPACE}_{lin}(A)$ as $\mathrm{PSPACE}_{lin}(A)$ but using nondeterministic machines, we have:

Proposition 1. For every oracle A, $\mathrm{PSPACE}_{lin}(A) = \mathrm{NPSPACE}_{lin}(A)$.

As shown in [5], this is false if the length of queries is not bounded explicitly (i.e, is bounded only by the exponential running time of polynomial space machines). The proof of Proposition 1 is a routine adaptation of the proof that PSPACE = NPSPACE (see, for instance, Chapter 2 of [1]).

On the other hand, unrestricted E machines can query strings of exponential length that are inaccessible to PSPACE_{lin} machines. Hence, if we wish to compare PSPACE to E in a fair way, we must also impose a linear bound on the length of queries made by E machines. Let $\mathrm{E}_{lin}(A)$ denote the class of sets accepted by E machines with this restriction, relative to oracle A. Our main result is as follows:

Theorem 2.

(1) $\mathrm{P} \subset \mathrm{PLOG}$ if and only if $\mathrm{E}_{lin}(A) \subset \mathrm{PSPACE}_{lin}(A)$ for every oracle A.

(2) $\mathrm{PLOG} \subset \mathrm{P}$ if and only if $\mathrm{PSPACE}_{lin}(A) \subset \mathrm{E}_{lin}(A)$ for every oracle A.

The proof is given in the next section. Let us note that part (1) of the theorem still holds if we replace the linear bound with a polynomial bound. The relativization of PSPACE is more natural in this case because the same bound applies to the query tape and the work tapes.

In Section 3 we present an application of our proof technique to other pairs of classes.

2. Proof of Theorem 2

All our languages are defined over the alphabet $\{0,1\}$. We assume some easy isomorphism between strings in $\{0,1\}^\star$ and natural numbers, and use them interchangeably. For a language A and a natural number n, $A^{\leq n}$ denotes $\{x \mid x \in A \wedge |x| \leq n\}$.

We associate to each string $o \in \{0,1\}^\star$ a *finitely defined oracle* in the natural way: the value of the oracle for the ith string in $\{0,1\}^\star$ is the ith bit of o. When no ambiguity is possible we identify string o with the oracle it encodes.

For an oracle A and a machine M, $L(M, A)$ is the language accepted by M relative to A. If A is finitely defined, $L(M, A)$ is defined only if all the queries in the computation are defined by A.

Part (1), from left to right.

Let E be a fixed E_{lin} machine that on inputs of length n runs in time 2^{cn} and makes queries of length at most cn. Define

$$\mathcal{OR}(E) = \{\, \langle x, o\rangle \mid |o| = 2^{c|x|+1} - 1 \wedge E^o(x) \text{ accepts} \,\}$$

where E^o denotes the computation of E relative to the oracle encoded in o. Note that every possible query of $E(x)$ can be answered from o.

It is easy to see that $\mathcal{OR}(E)$ can be decided in polynomial time. Assuming that $\mathrm{P} \subset \mathrm{PLOG}$, there is a machine M_1 that decides $\mathcal{OR}(E)$ using workspace $\log^j n$, for some j.

Consider now the following oracle machine M_2:

> On input x of length n, simulate M_1 with input $\langle x, o\rangle$, where $|o| = 2^{cn+1} - 1$ and o is never fully constructed. Instead, when the ith bit of o is needed, query i to the oracle and return the answer (0 or 1) as the required symbol of o.

For every oracle A and string x, with $|x| = n$, $M_2(x)$ accepts relative to A if and only if $\langle x, o\rangle$ is in $\mathcal{OR}(E)$, where o encodes $A^{\leq cn}$. And this happens if and only if $E(x)$ accepts relative to A. Hence, relative to any oracle A, $L(M_2, A) = L(E, A)$. Furthermore, M_2 uses space $\log^j |\langle x, o\rangle| = \log^j(2^{O(cn)}) = |x|^{O(1)}$, and makes queries of length at most cn. Therefore, for any oracle A, $L(E, A)$ is in $\mathrm{PSPACE}_{lin}(A)$. ∎

Part (1), from right to left.

For this part we need the following lemma. Informally, it states that if a machine N always accepts PSPACE_{lin} languages, then we can fix a single PSPACE_{lin} machine that is almost equivalent to N.

Lemma 3. Let N be a Turing machine that halts on every input and relative to any oracle, and assume that

$$\forall A \quad \exists M_A, \text{ a PSPACE}_{lin} \text{ machine, such that } L(N, A) = L(M_A, A).$$

Then

$$\exists M, \text{ a PSPACE}_{lin} \text{ machine}, \quad \exists B, \text{ a finitely defined oracle, such that}$$

$$\forall A, \text{ extension of } B,\ L(N, A) = L(M, A).$$

Proof. Suppose that the second statement in the lemma does not hold for N, that is, for every PSPACE$_{lin}$ machine M and every finitely defined oracle B, we can extend B so that M and N do not accept the same language under the extended oracle. We will show that the first statement in the lemma cannot hold.

Let A be the oracle constructed as follows: Oracle A_0 is undefined for every string. For every $k > 0$, let M_k be the kth PSPACE$_{lin}$ machine; let A_k be a minimal, finitely defined oracle that extends A_{k-1} and such that $L(N, A_k) \neq L(M_k, A_k)$. With our hypothesis, we can always find such an extension. Define A to be $\bigcup_{k>0} A_k$. Clearly, no PSPACE$_{lin}(A)$ machine accepts $L(N, A)$, so A witnesses that the first statement is false. ■ *Lemma*

Now consider any language L_0 in P. It is easily seen that L_0 is logspace-equivalent to the padded language

$$L = \{\, x10^k \ \mid\ x \in L_0 \text{ and } |x10^k| = n + 2^n \text{ for some } n \,\}.$$

That is, every string in L has length $n+2^n$ for some n. To prove this part of the theorem, we will show that L is in PLOG assuming that E$_{lin}(A) \subset$ PSPACE$_{lin}(A)$ relative to every A.

Let P be a machine that accepts L in polynomial time. Let E be the following oracle machine:

> On input u of length n, query strings $|u| + 1,\ |u| + 2,\ \ldots,\ |u| + 2^n$ to the oracle and interpret the answers as consecutive bits of a string v of length 2^n; then, simulate $P(uv)$.

It is easy to show that E runs in time $2^{O(n)}$ (with the constant in the exponent not depending on the oracle queried), and that it makes queries of linear length; therefore, for every oracle A, $L(E, A)$ belongs to E$_{lin}(A)$. By hypothesis we have that $L(E, A) \in$ PSPACE$_{lin}(A)$ for every A, that is, for each oracle A there is a PSPACE$_{lin}$ machine M_A such that $L(E, A) = L(M_A, A)$.

By Lemma 3, we can find *one* PSPACE_{lin} machine M that accepts $L(E, A)$ relative to any oracle A extending a certain finitely defined oracle B. We use M to give an algorithm for L.

For any two strings u and v, let $\text{Pad}(u, v)$ be the oracle that answers query y as follows:

- if B is defined for y, answer as B does;
- otherwise, if number y is in $\{|u| + 1, \ldots, |u| + |v|\}$, answer bit number $y - |u|$ of v;
- otherwise, answer 0.

And now define machine P':

> On input x of length $n + 2^n$, split x into strings u and v, $x = uv$, with $|u| = n$ and $|v| = 2^n$; simulate $M(u)$ relative to $\text{Pad}(u, v)$.

When x is sufficiently long, $\text{Pad}(u, v)$ is an extension of B, so P' accepts uv if and only if $u \in L(M, \text{Pad}(u, v)) = L(E, \text{Pad}(u, v))$. By the definition of E, this is true if and only if $P(uv)$ accepts. Therefore, P' decides L correctly except for a finite number of strings (namely, strings $x = uv$ with u small with respect to B).

Finally, recall that some polynomial p bounds the space used by M under every oracle. Hence, P' on input uv uses space $p(|u|) \leq p(\log |uv|)$. This implies that $L \in \text{PLOG}$. ■

Part (2), from left to right.

Let M be a fixed PSPACE_{lin} machine that on inputs of length n uses space n^c and makes queries of length at most cn. Define

$$\mathcal{OR}(M) = \{ \langle x, o \rangle \mid |o| = 2^{c|x|+1} - 1 \wedge M^o(x) \text{ accepts} \}$$

It is easy to see that $\mathcal{OR}(M)$ can be decided in space $O(|x|^c + c|x|)$, that is, polylog in $|\langle x, o \rangle|$. Assuming that $\text{PLOG} \subset \text{P}$, there is a machine M_1 that decides $\mathcal{OR}(M)$ in polynomial time.

Consider now the following oracle machine E:

> On input x and relative to A, query A to obtain the string o of length $2^{c|x|+1} - 1$ that encodes $A^{\leq c|x|}$; then simulate M_1 with input $\langle x, o \rangle$.

By an argument as in part (1), $L(E, A) = L(M, A)$ for every A. Furthermore, on an input x, E runs in time polynomial in $|\langle x, o \rangle|$, that is, $2^{O(|x|)}$. Therefore, for any oracle A, $L(M, A)$ is in $\text{E}_{lin}(A)$. ■

Part (2), from right to left.

Consider any language $L_0 \in$ PLOG. It is easily seen that L_0 is logspace-equivalent to the padded language

$$L = \{\, x10^k \mid x \in L_0 \text{ and } |x10^k| = n + 2^n \text{ for some } n \,\}.$$

We will show that L is in P assuming that $\mathrm{PSPACE}_{lin}(A) \subset \mathrm{E}_{lin}(A)$ relative to every A.

Let M be a machine that accepts L in polylog space. Without loss of generality we make the following assumptions about M: Its input tape is read-only and infinite to the right, with cells numbered 1, 2, ... An input of length n is given to M in cells 1 to n, with all other cells containing blanks. M never moves its input head beyond the first blank cell in the input tape. Now let M_1 be the following oracle machine:

> On input u of length n, simulate M on an input of length $n + 2^n$ as follows: Worktapes of M are simulated in worktapes of M_1. When the simulation needs the ith symbol in the input tape of M,
>
> – use the ith symbol of u, if $1 \le i \le n$;
>
> – query string i to the oracle and use the answer, if $n < i \le n + 2^n$;
>
> – otherwise (i.e., if $i = n + 2^n + 1$), assume a blank symbol is read.

Since M uses polylog space, M_1 uses space $\log^{O(1)}(n+2^n)$, or polynomial in n, and makes queries of linear length. Therefore, for every oracle A, $L(M_1, A) \in \mathrm{PSPACE}_{lin}(A)$. By hypothesis we have that $L(M_1, A) \in \mathrm{E}_{lin}(A)$ for every A, that is, for each oracle A there is an E_{lin} machine E_A such that $L(M_1, A) = L(E_A, A)$.

Arguing as in Lemma 3, we can fix an E_{lin} machine E that accepts $L(M_1, A)$ relative to any oracle A extending a certain finitely defined oracle B. Define oracle $\mathrm{Pad}(u, v)$ as in the proof of part (1). We use E to give an algorithm for L.

> On input x of length $n + 2^n$, split x into strings u and v, $x = uv$, with $|u| = n$ and $|v| = 2^n$; accept if and only $E(u)$ accepts relative to $\mathrm{Pad}(u, v)$.

This algorithm runs in time $2^{O(|u|)} = |uv|^{O(1)}$. Also, when x is sufficiently long, $\mathrm{Pad}(u, v)$ extends B. Therefore the algorithm accepts $x = uv$ if and only if $u \in L(E, \mathrm{Pad}(u, v)) = L(M_1, \mathrm{Pad}(u, v))$, and this is equivalent to $uv \in L(M) = L$ by the definition of M_1. Hence, L is in P. ■

3. An application to other classes

The proof of Theorem 2 is based on translations of inclusions between higher and lower classes. Similar techniques had been used before to relate inclusions of exponential-time and polynomial-time classes.

Let NE stand for the class of languages decided by nondeterministic machines running in time $2^{O(n)}$. It is known that $\mathrm{E} = \mathrm{NE}$ if and only if there are no tally sets in $\mathrm{NP} - \mathrm{P}$ [3], and if and only if there are no sparse sets in $\mathrm{NP} - \mathrm{P}$ [6]. As a direct consequence, $\mathrm{P} = \mathrm{NP}$ implies $\mathrm{E} = \mathrm{NE}$. It is not known, however, whether $\mathrm{E} = \mathrm{NE}$ implies $\mathrm{P} = \mathrm{NP}$. The strongest result known so far is that this implication is false relative to some oracles [9].

With the technique in the previous section we can show a relation of a different kind between polynomial time and exponential time. Recall that a tally set is any subset of $0^\star$.

Theorem 4. The following three statements are equivalent.

(a) $\mathrm{P} = \mathrm{NP}$.

(b) For every tally set T, $\mathrm{P}(T) = \mathrm{NP}(T)$.

(c) For every tally set T, $\mathrm{E}(T) = \mathrm{NE}(T)$.

That is, the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ question is equivalent to its relativization to tally sets and to the relativization of $\mathrm{E} \stackrel{?}{=} \mathrm{NE}$ to tally sets, although it may not be directly equivalent to unrelativized $\mathrm{E} \stackrel{?}{=} \mathrm{NE}$.

Observe that, in this theorem, E and NE are relativized in the standard way, with no restriction on the length of queries. It is the quality of oracles that we restrict here.

Sketch of proof. Long proved the equivalence of (a) and (b) ([8], Lemma 3.5). (b) implies (c) because the proof sketched above for "$\mathrm{P} = \mathrm{NP}$ implies $\mathrm{E} = \mathrm{NE}$" relativizes to any oracle.

And (c) implies (a) with a proof like those for the right to left directions of Theorem 2. Let L be a language in NP padded so that every string in L has length $n + 2^n$. For every tally set T, define

$$\mathcal{L}(T) = \{\, u \mid u\mathcal{X}_T(|u|+1)\ldots\mathcal{X}_T(|u|+2^{|u|}) \in L \,\}$$

were $\mathcal{X}_T(i)$ is the value of the characteristic function of T applied to string 0^i.

For every T, $\mathcal{L}(T)$ is in $\mathrm{NE}(T)$. By hypothesis, for every T there is some E machine that accepts $\mathcal{L}(T)$ relative to T. As in Lemma 3, there must exist a fixed E machine M and a finitely defined tally oracle T_0 such that for every tally set T extending T_0, M accepts $\mathcal{L}(T)$ relative to T.

As in Theorem 2, we can now build another machine that splits its input into two parts. The first part is used as input to M. The second part is exponentially longer than the first one, and is used to answer oracle queries. This machine accepts a finite variation of L in polynomial time. ■

Acknowledgements

I thank the three anonymous referees for many helpful comments. In particular, one of them suggested to look for part (2) of the main theorem. I am also grateful to Lane Hemachandra for his help during the reviewing process.

References

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

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

[3] R. Book: "Tally languages and complexity classes". *Information and Control* **26** (1974), 186–193.

[4] R. Book: "Translational lemmas, polynomial time, and $(\log n)^j$-space". *Theoretical Computer Science* **1** (1976), 215–226.

[5] R. Gavaldà, L. Torenvliet, O. Watanabe, and J.L. Balcázar: "Generalized Kolmogorov complexity in relativized separations". In *Proceedings of the 15th Symposium on Mathematical Foundations of Computer Science*, 269–276. Lecture Notes in Computer Science 452. Springer-Verlag 1990.

[6] J. Hartmanis: "On sparse sets in NP−P". *Information Processing Letters* **16** (1983), 55–60.

[7] J. Hartmanis: "New developments in structural complexity theory". *Proceedings of the 15th International Colloquium on Automata, Languages, and Programming*, 271–286. Lecture Notes in Computer Science 317. Springer-Verlag 1988.

[8] T. Long: "On restricting the size of oracles compared with restricting access to oracles". *SIAM Journal on Computing* **14** (1985), 585–597.

[9] C. Wilson: "Relativization, reducibilities and the exponential hierarchy". M. Sc. Thesis. Technical Report 140/80, Department of Computer Science, University of Toronto, 1980.