Measuring in PSPACE*

Elvira Mayordomo
Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo 5
08028 Barcelona, Spain
E-mail: `mayordomo@lsi.upc.es`

Abstract

Results of the kind "Almost every oracle in exponential space separates P from NP" or "Almost every set in exponential time is P-bi-immune" can be precisely formulated via a new approach in Structural Complexity recently introduced by Lutz. He defines a resource-bounded measure in exponential time and space classes that generalizes Lebesgue measure, a powerful mathematical tool.

We investigate here the possibility of extending this resource-bounded measure to other classes, mainly PSPACE. We prove here that the natural candidate of a resource bound for measuring in PSPACE is not valid unless some unlikely consequences are true. We then obtain a weaker way of measuring in PSPACE that lacks a property that resource-bounded measure has in bigger classes.

1 Introduction

Resource-bounded measure was introduced by Lutz in [3] and [4]. It deals with complexity classes within exponential time or space, distinguishing between "large" and "small" classes.

This method generalizes a powerful mathematical tool, Lebesgue classical measure, which is useless when dealing with recursive classes since it gives measure zero to all of them. Resource-bounded measure limits the time or space that can be used to compute the measure of a class, so that it is possible to have "large" (measure 1) recursive classes.

Many of the results that have been obtained up to now in this field ([4], [5], [6], [7]) are based on an existence result. For instance, the fact that there exist a P-bi-immune language in exponential time ([2]) can be extended to prove that

*This work was supported by a Spanish Government grant FPI PN90 and by the ESPRIT EC project 3075 (ALCOM).

almost every language in exponential time is P-bi-immune ([7]). Such extensions give a more general idea about what the typical behavior of the languages in the class of exponential time is. We want to extend some results concerning existence in PSPACE to "almost every", for instance, the fact that there exists a language in PSPACE that is not DLOG-self-reducible.

To introduce his resource-bounded-measure, Lutz takes a constructive way of defining Lebesgue measure (by covers in [3] and with density functions in [4]) and bounds the resources used in the construction to obtain meaningful measures in exponential classes, that is to say, measures that allow existence of both small and big subclasses. For exponential time the correct bound is polynomial time and for exponential space the correct bound is polynomial space.

In this line, it would be a natural solution for our problem of PSPACE to pose a restriction of poly-logarithmic space to obtain a meaningful measure within PSPACE, but the sublinearity of this bound poses some technical difficulties, which are not easily solvable since this would imply some unlikely results. We then impose more restrictions and obtain a meaningful "measure" in PSPACE. The properties of this measure are not as good as in bigger classes.

2 Preliminaries

We will use the alphabet $\Sigma = \{0,1\}$. A string is a finite sequence $x \in \Sigma^*$. We write $|x|$ for the length of x. The unique string of length 0 is λ, the empty string. A sequence is an infinite sequence $x \in \Sigma^\infty$.

If x is a string and y is a string or sequence, then xy is the concatenation of x and y. If x is a string and $k \in \mathbf{N} \cup \{\infty\}$, then x^k is the k-fold concatenation of x with itself.

Let $s_0, s_1, s_2, ...$ be the standard enumeration of the strings in Σ^* in lexicographical order.

From now on we will use the bitstring χ_L of a language L to denote it, where χ_L is defined as follows:

$$\chi_L \in \Sigma^\infty \text{ and } \chi_L[i] = 1 \text{ iff } s_i \text{ belongs to } L.$$

So we identify the class $\mathcal{P}(\Sigma^*)$ of all languages over Σ with the set Σ^∞ of all sequences. The complement of a set of languages X is $X^c = \Sigma^\infty - X = \mathcal{P}(\Sigma^*) - X$.

If x is a string and y is a string or sequence, then $x \sqsubseteq y$ iff there exists a string or sequence z such that $y = xz$, and $x \sqsubsetneq y$ iff $x \sqsubseteq y$ and $x \neq y$.

Definition 1 *Let $w \in \Sigma^*$. C_w denotes the class of languages $\{x \in \Sigma^\infty \mid w \sqsubseteq x\}$.*

Let *all* be the class of all functions $f : \Sigma^* \to \Sigma^*$, and rec be the class of recursive functions in *all*. We fix once and for all a one-to-one pairing function $\langle , \rangle$

from $\Sigma^* \times \Sigma^*$ onto Σ^* such that the pairing function and its associated projections, $\langle x, y\rangle \mapsto x$ and $\langle x, y\rangle \mapsto y$ are computable in linear time and logarithmic working space.

We let $\mathbf{D}= \{m2^{-n} \mid m, n \in\mathbf{N}\}$ be the set of *nonnegative dyadic rationals.*

With the exception of functions mapping into $[0, \infty)$ all our functions are of the form $f : X \to Y$, where each of the sets X, Y is $\mathbf{N}, \Sigma^*, \mathbf{D}$, or some cartesian product of these sets. For purposes of computational complexity we regard such functions as mapping Σ^* into Σ^*. For example, a function $f : \mathbf{N}^2 \times \Sigma^* \to \mathbf{N} \times \mathbf{D}$ is interpreted as a function $\tilde{f} : \Sigma^* \to \Sigma^*$. Under this interpretation, $f(i, j, w) = (k, q)$ means that $\tilde{f}(\langle 0^i, \langle 0^j, w\rangle\rangle) = \langle 0^k, \langle u, v\rangle\rangle$, where u and v are the binary representations of the integer and fractional parts of q, respectively.

For a function $f : \mathbf{N}\times X \to Y$ and $k \in \mathbf{N}$, we define the function $f_k : X \to Y$ by $f_k(x) = f(k, x)$. For a function $f : \mathbf{N}^n \times X \to Y$, we write $f_{k,l} = (f_k)_l$, etc. For a function $f : \Sigma^* \to \Sigma^*$, we write f^n for the n-fold composition of f with itself.

Let RE be the class of recursively enumerable languages, and REC be the class of recursive languages.

Let E be the class $\bigcup_c \mathrm{DTIME}(2^{cn})$, p be the class of polynomial time computable functions, and pspace be the class of polynomial space computable functions.

We will use Δ in the rest of the paper to refer to a class of functions.

Next, we can associate with each class of functions Δ a class of languages $\mathrm{R}(\Delta)$ as follows:

Definition 2 *$f \in \Delta$ is a constructor iff $\forall w \in \Sigma^*, w \sqsubsetneq f(w)$.*

Definition 3 *Let h be a constructor in Δ, then $\mathrm{R}(h)$ is the unique element in Σ^∞ such that $\forall i \ h^i(\lambda) \sqsubseteq \mathrm{R}(h)$.*

Definition 4 *$\mathrm{R}(\Delta)$ is the class of languages $\{\mathrm{R}(h) \mid h$ a constructor in $\Delta\}$.*

From the classes of functions we mentioned, we obtain well-known classes [3]:

$\mathrm{R}(\mathrm{p}) = \mathrm{E}$,
$\mathrm{R}(\mathrm{pspace}) = \mathrm{ESPACE}$,
$\mathrm{R}(\mathrm{rec}) = \mathrm{REC}$,
$\mathrm{R}(all) = \mathcal{P}(\Sigma^*)$.

3 Resource-bounded measure

In this section we introduce resource-bounded measure by Lutz [4] for a general resource bound Δ. We will use Δ for both a resource bound and the class of recursive functions defined by it.

We will remark that one of the properties of this resource-bounded measure, namely Δ-additivity (Lemma 1, from [4]) does not necessarily hold for certain resource bounds. For it to hold we will assume the resource bound to be closed under linear sum, as defined next.

Definition 5 *A class of functions Δ is closed under linear sum iff for every function $f \in \Delta$, $g(w) = \sum_{i=0}^{|w|} f_i(w)$ is in Δ.*

First we introduce the concept of density function, basic in this theory.

Definition 6 *A density function is a function $d : \Sigma^* \to [0, \infty)$ satisfying*

$$d(w) \geq \frac{d(w0) + d(w1)}{2}$$

for all $w \in \Sigma^*$. The global value of a density function d is $d(\lambda)$. The set covered by a density function d is

$$\mathrm{S}[d] = \bigcup_{\substack{w \in \Sigma^* \\ d(w) \geq 1}} C_w.$$

Definition 7 *A density function d covers a set $X \subset \Sigma^\infty$ if $X \subset \mathrm{S}[d]$.*

Consider the random experiment in which a sequence $x \in \Sigma^\infty$ is chosen by using an independent toss of a fair coin to decide each bit of x. Definition 6 implies that $\Pr[x \in \mathrm{S}[d]] \leq d(\lambda)$. Intuitively, we will regard a density function d as a verification that $\Pr[x \in X] \leq d(\lambda)$ for all sets $X \subset \mathrm{S}[d]$.

More generally, we will be using "uniform systems" of density functions that are Δ-computable.

Definition 8 *An n-dimensional density system (n-DS) is a function*

$$d : \mathbf{N}^n \times \Sigma^* \to [0, \infty)$$

such that $d_{\vec{k}}$ is a density function for every $\vec{k} \in \mathbf{N}^n$.

Definition 9 *An n-DS d is Δ-computable iff there is a function $\hat{d} \in \Delta$, such that*

$$\hat{d} : \mathbf{N}^{n+1} \times \Sigma^* \to \mathbf{D}, \quad |\hat{d}_{\vec{k},r}(x) - d_{\vec{k}}(x)| \leq 2^{-r}$$

for all $\vec{k} \in \mathbf{N}^n, r \in \mathbf{N}$, and $x \in \Sigma^*$.

We now come to the key idea of resource-bounded measure theory.

Definition 10 *A set $X \subset \Sigma^\infty$ has Δ-measure 0 iff there exists a Δ-computable 1-DS d such that, for all $k \in \mathbf{N}$, d_k covers X with global value $d_k(\lambda) \leq 2^{-k}$.*

Thus a set X has Δ-measure 0 if Δ provides sufficient computational resources to compute uniformly good approximations to a system of density functions that cover X with rapidly vanishing global value.

Definition 11 *A set $X \subset \Sigma^{\infty}$ has Δ-measure 1 iff X^c has Δ-measure 0.*

The "typical" languages in this formulation are called Δ-random.

Definition 12 *A language $L \in \Sigma^{\infty}$ is Δ-random iff it belongs to every Δ-measure 1 class.*

We are interested in measuring inside $\mathrm{R}(\Delta)$.

Definition 13 *A set $X \subset \Sigma^{\infty}$ has measure 0 in $\mathrm{R}(\Delta)$ iff $X \cap \mathrm{R}(\Delta)$ has Δ-measure 0.*

Definition 14 *A set $X \subset \Sigma^{\infty}$ has measure 1 in $\mathrm{R}(\Delta)$ iff X^c has measure 0 in $\mathrm{R}(\Delta)$.*

The first thing to prove is that we have defined a measure with some meaning in $\mathrm{R}(\Delta)$, that is to say, there exist $X \subseteq \Sigma^{\infty}$ such that X does not have measure 0 in $\mathrm{R}(\Delta)$. This is proved by Theorem 3.13 in [4] (Measure Conservation Theorem), stated as

Theorem 1 *If $w \in \Sigma^*$ and d is a Δ-computable density function that covers $C_w \cap \mathrm{R}(\Delta)$, then $d(\lambda) \geq 2^{-|z|}$.*

It is clear then that C_w does not have measure 0 in $\mathrm{R}(\Delta)$.

Next we will examine two basic properties of resource-bounded measure. The following lemma from [4] states that a special kind of countable union of Δ-measure 0 sets has measure 0.

Definition 15 *A set X is a Δ-union of the Δ-measure 0 sets $X_0, X_1, X_2, \ldots$ iff $X = \bigcup_{j=0}^{\infty} X_j$ and there exists a Δ-computable 2-DS d such that each d_j witnesses that X_j has Δ-measure 0.*

Lemma 1 *If Δ is a class of functions closed under linear sum and X is a Δ-union of Δ-measure 0 sets, then X has Δ-measure 0.*

Lemma 2 is another straightforward property.

Lemma 2 *Let $X,\ Y \subset \Sigma^{\infty}$. If $Y \subset X$ and X has Δ-measure 0, then Y has Δ-measure 0.*

Lemmas 1 and 2 are basic properties that any complete measure, as defined in Mathematics, must have. For us these two properties will be very helpful when proving that a certain set has measure 0 ([4], [5], [7]).

4 Measure in PSPACE

In this section we will examine two different possible measures for PSPACE. In order to have a meaningful measure in PSPACE, the candidates are classes of functions Δ such that R(Δ) =PSPACE, because Theorem 1 tells us that for any resource bound Δ satisfying this equation, PSPACE does not have Δ-measure 0.

By analogy to the resource bounds we have posed for E and ESPACE, respectively polynomial time and polynomial space, the corresponding resource bound for PSPACE would be poly-logarithmic space and we would like to prove that, if polylogspace is the set of functions computable with working space poly-logarithmic in the size of the input, then R(polylogspace)=PSPACE.

The following lemma proves the first part of this equality.

Lemma 3 PSPACE$\subseteq$R(polylogspace).

Proof Given L a language in PSPACE, we have to define a constructor h such that $\mathrm{R}(h) = L$. We do it as follows $h(w) = wL(s_{|w|})$, where $L(y)$ is 1 if $y \in L$, 0 otherwise. It is straightforward to check that h is in polylogspace. □

The other part is more complicated, since to recognise an input $x = s_i$ in a language $\mathrm{R}(h)$ we have to simulate the sucessive computations λ ,$h(\lambda)$, $h(h(\lambda))$, $h(h(h(\lambda)))$, etc. until $|h^j(\lambda)| > i$. But the output of many of these sucessive computations will be too big to be kept in space $|x|^k$, for x big enough.

Another approach would be to simulate the computations as before but without writing the full output, that is, recalculating the bits in $h^m(\lambda)$ needed in the computation of $h^{m+1}(\lambda)$. But even in this case the stack of the recursion can be too big for PSPACE (e.g. in the cases where $|h(w)| = |w| + 1$ for every input).

We see in the next theorem what is really R(polylogspace). It corresponds to a class of self-reducible languages that is expected to be different from PSPACE.

Definition 16 *A language A on Σ is* PSPACE-*wdq-self-reducible (wdq stands for word decreasing queries) if $A = L(M, A)$, where M is a* PSPACE *machine that makes only queries strictly smaller than the input (in lexicographical order).*

Theorem 2 R(polylogspace) *is exactly the class of* PSPACE-*wdq-self-reducible languages.*

Proof If h is a constructor in polylogspace, then it is easy to see if $z = s_i$ is in $\mathrm{R}(h)$ by simulating the succesive computations $h(\lambda)$, $h(h(\lambda))$, $h(h(h(\lambda)))$,... until $|h^k(\lambda)| > i$, and every time we want to read a bit of the input we make a question to $\mathrm{R}(h)$ instead. This takes polynomial space and the questions are always lexicographically smaller than z, so $\mathrm{R}(h)$ is clearly PSPACE-wdq-self-reducible.

In the other direction, given L a PSPACE-wdq-self-reducible language via a Turing Machine M, we can define h as in Lemma 3:

$h(w) = wL(s_{|w|})$, where $L(y)$ is 1 if $y \in L$, 0 otherwise.

To decide if $s_{|w|}$ is in L, we can use that L is PSPACE-wdq-self-reducible, and simulate the computation of M on $s_{|w|}$ answering to a query s_i, $(i < |w|)$ by checking the ith bit of w. This simulation can be done in space polynomial in the length of $s_{|w|}$, that is, poly-logarithmic in the length of w, so h is in polylogspace. □

The definition of PSPACE-wdq-self-reducible is similar to P-wdq-self-reducible sets in [1]. The only difference is that in [1] the machines considered were in P. In this article, Balcázar proved that E has $\leq_m^p$-complete languages that are P-wdq-self-reducible. Since every P-wdq-self-reducible language is clearly PSPACE-wdq-self-reducible, we have the following result.

Theorem 3 *If* PSPACE=R(polylogspace) *then* E $\subseteq$ PSPACE, *and* EXP = PSPACE, *where* EXP = $\bigcup_k$ DTIME(2^{n^k}).

Proof Balcazar proved in [1] that E has a PSPACE-wdq-self-reducible language A (see the comment above). In the hypothesis that PSPACE=R(polylogspace) and using Theorem 2 it is clear that A is in PSPACE.

Since PSPACE is closed under $\leq_m^p$-reduction, if PSPACE contains A, it contains the whole class E.

To finish our proof, we notice that by an easy padding argument, every E-complete problem is EXP-complete, and that PSPACE $\subseteq$ EXP, so in this case EXP=PSPACE. □

There are other restrictions we can pose on polylogspace functions still being able to construct the full PSPACE. For instance we can consider only functions that can be computed with on-line polylogspace machines, that is to say, machines that read the input only once and from right to left.

Definition 17 *Let* plogon *be the class of functions that are computable by on-line machines with working space polylogarithmic in the size of the input.*

Theorem 4 R(plogon)=PSPACE.

Proof For right to left, we can use the constructor in Lemma 3, which can be clearly computed with an on-line machine.

In the other direction, given h in plogon we can see if $z = s_i$ is in R(h) by simulating the succesive computations $h(\lambda)$, $h(h(\lambda))$, $h(h(h(\lambda)))$,... by couples. Since we are using an on-line machine, the computation of $h^l(\lambda)$ is identical to the one of $h^{l+1}(\lambda)$, until $h^l(\lambda)$ finds the end of its input. We take advantage of this to simulate two computations in parallel as follows:

Begin

$l := 1$, $j := 0$

C :=initial configuration in the computation of $h(\lambda)$

While $j \leq (i+1)$ do

simulate $h^l(\lambda)$ starting in C until getting the jth bit of the output.

$b :=$ the mentioned jth bit.

If b is not blank then

simulate $h^{l+1}(\lambda)$ starting in C until reading the jth bit of the input.

C :=last configuration reached in the simulation.

$j := j+1$

else $l := l+1$

endif

endwhile

$z \in$R(h) iff $b = 1$.

End

In this way we do not need to keep long outputs, and these simulations can take place in polynomial space in the length of z. $\square$

The problem of the class plogon is that it does not seem easy to prove that it is closed under linear sum, so we cannot be sure that the measure defined by this bound has the property of Δ-additivity (Lemma 1) that makes it a measure in the classical sense, and gives us a useful tool for new results.

5 Conclusion

We have a way of measuring in PSPACE, different from the intuitive approach that provably does not hold.

The open problem is to find some nice properties of this measure, that allow us to prove results in the line of the existing ones for E and ESPACE. We are now working on finding a more restrictive type of Δ-union that satisfies Lemma 1 for the class Δ =plogon.

J. L. Balcázar proposed still another approach for a measure in PSPACE that probably deserves further attention. It is based in considering only those constructors whose output has at least twice the length of the input. This kind of "honest" constructors give us exactly PSPACE, and Lemma 1 (Δ-additivity) holds, but the problem is to prove that this is non-trivial, that is, that PSPACE does not have measure 0.

Acknowledgements

I would like to thank José Luis Balcázar, who proposed this problem and gave a lot of ideas and suggestions and Jack Lutz, who explained to me the intuition behind his theory.

References

[1] J.L. Balcázar: Self-Reducibility. *Journal of Computer and System Sciences* **41** (1990), 367–388.

[2] J. Hartmanis, L. Berman: On Isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing* **6** (1977), 305–332.

[3] J.H. Lutz: Category and Measure in Complexity Classes *SIAM Journal on Computing* **19** (1990), 1100–1131.

[4] J.H. Lutz: Almost Everywhere High Nonuniform Complexity. *Journal of Computer and System Sciences* **44** (1992), 220–258.

[5] J.H. Lutz, E. Mayordomo: Measure, Stochasticity and the Density of Hard Languages. *SIAM Journal on Computing,* to appear.

[6] J.H. Lutz, W. Schmidt: Circuit Size Relative to Pseudorandom Oracles. In: *Proceedings 5th Annual Conference on Structure in Complexity Theory* (1990), 268–286.

[7] E. Mayordomo: Almost every set in exponential time is P-bi-immune. *Theoretical Computer Science,* to appear.