CONNECTIONS AMONG NONUNIFORM MODELS FOR PROBLEMS REQUIRING HIGH AMOUNT OF RESOURCES

(Preliminary version, november 1985)

José L. Balcázar and Joaquim Gabarró
Facultat d'Informática de Barcelona
Universitat Politécnica de Catalunya
Barcelona 08034, SPAIN

Abstract: We characterize in terms of oracle Turing machines the classes defined by exponential lower bounds on some nonuniform complexity measures. After, we use the same methods to give a new characterization of classes defined by polynomial and polylog upper bounds, obtaining an unified approach to deal with upper and lower bounds. The measures are the initial index, the context-free cost, and the boolean circuits size. We interpret our results by discussing a trade-off between oracle information and computed information for oracle Turing machines.

I. Introduction.

There are two fundamental aproaches to complexity theory, known as "uniform" and "nonuniform". The uniform approach considers problems as a global object and measures the amount of resources needed to solve it using models of computation usually equivalent to Turing machines. The non-uniform approach considers finite approximations to the problem and measures the complexity of these approximations. This non-uniform approach uses computational models

adequate to the finiteness of the approximations. The most widely used model is the boolean circuit model [17]. Many other models of non-uniform complexity measures exists, as the initial index [9] or the context-free cost [4].

Many connections exist between uniform and nonuniform measures [18], [2], [1]. These papers are mainly devoted to the study of classes defined by small (for example polynomial) upper bounds on the uniform resources and/or the nonuniform complexity.

We investigate here the connection between uniform and non-uniform models for problems requiring a high amount of resources. Several interesting results can be shown about high lower bounds on different nonuniform models:

- Consider the initial index [9] (see definitions below). To prove exponential lower bounds for problems is not difficult: the two type parenthesis language has an exponentially growing initial index. The same happens with the set of palindroms over an alphabet of at least two letters. It is also known [19] that almost all languages in Σ^n have an initial index of

$$(1-\varepsilon)\frac{||\Sigma||^n}{n.\log||\Sigma||}$$

- Consider the context-free cost. For almost the languages in Σ^n a similar bound as the one above has been established in [10]. Another interesting property regarding context-free cost appears in [8]. It is shown there that the set of words containing an square is not context-free; actually, the proof shows that this set has exponential context-free complexity.

- Consider the boolean circuit complexity [17]. It is well known (see [11]) that almost all boolean functions of n variables need

$$(1-\varepsilon)\frac{2^n}{n}$$

gates to be synthesized.

This set of facts induce us to consider the classes defined by exponential lower bounds on these nonuniform measures. We give a characterization of these families in terms of oracle Turing machines. As usual (see [1]) the oracles are used to "break up" the uniformity. These results are presented in section III; section II is devoted to basic definitions and facts.

In section IV we give a new characterisation of the classes defined by upper bounds. This characterisation uses methods similar

to the methods employed in the previous section. Here we approach the problem from the viewpoint of the "quantity" of oracle as opposed to the approach of the "quality" of oracle taken in [1]. With this new characterization, lower and upper bounds adopt a very similar treatment and appear as dual forms of the same phenomenon.

Finally, we devote section V to the analysis of "trivial" nonuniform families, in which very small upper bounds are imposed on the Turing machine resources. We close this paper with a short section of conclusions and an appendix in which several facts and properties of the standard notations of orders of magnitude are shown. These facts are broadly used throughout the paper.

II. Preliminaries.

We assume the reader familiar with the basic concepts of Formal Language Theory. Problems are encoded as subsets of words over a fixed finite alphabet Σ consisting of at least two symbols. Hence a language is a subset of Σ^*. Computational models are finite automata, pushdown automata, and time or space bounded Turing machines, in their respective deterministic or nondeterministic versions. Relativized computations are performed by oracle Turing machines which query the oracle in one step about words written in a special oracle tape.

On-line machines are also used. In these machines the input tape head is only one-way, and the machine not allowed to read back its

input. For this model we assume that the length of the input is given to the machine in binary on a separate read-only tape. For undefined notions see [13].

We study the following nonuniform complexity measures:

- Given a language L, its initial index [9] is given by the function $a_L(n)$ whose value is the size (i.e., number of states) of the smallest nondeterministic finite automaton accepting exactly $L \cap \Sigma^n$.

- Given a language L, its context-free cost [4] is given by the function $cf_L(n)$ whose value is the size (i.e., number of rules) of the smallest context-free grammar accepting exactly $L \cap \leq^n$.

- Given a language L, its boolean size complexity or combinational complexity [17] is given by the function $c_L(n)$ whose value is the size (i.e., number of gates) of the smallest boolean circuit synthesizing the characteristic function of $L \cap \Sigma^n$.

Other similar functions, like branching program size [21], boolean formula size [17], and circuit depth [17], will be introduced in section IV.

III. Exponential lower bounds.

In this section we characterize the classes defined by

exponential lower bounds on the non-uniform complexity of the languages. Our notation for orders of magnitude follows [20]. In the appendix we present some useful facts about them.

Let m be a nonuniform complexity measure associating to every language L a function m_L. The family of sets having <u>exponential lower bound on m</u> is:

$$\Omega_\infty(\exp)[m] = \{ L \;/\; m_L(n) \in \Omega_\infty(\exp) \}$$

Examples of measures m are the initial index, the context-free cost, and the boolean cost, which were defined in section II. We can also define classes by considering subexponential upper bounds: the family of sets having <u>subexponential upper bound on m</u> is

$$o(\exp)[m] = \{ L \;/\; m_L(n) \in o(\exp) \}$$

From the facts shown in the appendix about $\Omega_\infty()$ and $o()$ notations follows easily that:

Lemma 1.

(1) $\Omega_\infty(\exp)[m] \cup o(\exp)[m] = P(\Sigma^*)$

(2) $\Omega_\infty(\exp)[m] \cap o(\exp)[m] = \emptyset$

In order to give the characterisation in terms of Turing machines we introduce some simple notation. Consider the oracle Turing machine model; it presents two very different memory devices: work tapes and oracle tape. A bound on the work space available imposes a limitation on the calculation capabilities of the machine, whereas a bound on the oracle tape imposes a bound on the amount of external help obtained by the machine. We denote:

$$\forall\ (\mathrm{Space}(\Omega_\infty(n)) \vee \mathrm{Oracle}(\Omega_\infty(n)))$$

the class of all sets L such that for every machine M and oracle A with L = L(M,A), either the work space or the oracle tape used by M are in $\Omega_\infty(n)$. Dually, we denote

$$\exists\ (\mathrm{Space}(o(n)) \wedge \mathrm{Oracle}(o(n)))$$

the class of all sets L such that there are a machine M and an oracle A with L = L(M,A), such that both the work space and the oracle tape of M are bounded by functions in o(n).

The following easy lemma might be called "duality principle". We leave the proof to the reader.

Lemma 2. Given any set L,

$$L \in \forall\,(\mathrm{Space}(\Omega_\infty(n)) \vee \mathrm{Oracle}(\Omega_\infty(n))) \text{ iff}$$

$$L \notin \exists\,(\mathrm{Space}(o(n)) \wedge \mathrm{Oracle}(o(n)))$$

Now we prove the relationship between exponential initial index and the oracle Turing machine classes as given in the following theorem.

Theorem 3. For nondeterministic on-line oracle Turing machines,

(1) $\Omega_\infty(\exp)[a] = \forall\,(\mathrm{Space}(\Omega_\infty(n)) \vee \mathrm{Oracle}(\Omega_\infty(n)))$

(2) $o(\exp)[a] = \exists\,(\mathrm{Space}(o(n)) \wedge \mathrm{Oracle}(o(n)))$

Proof.

We start by proving (2). Then (1) follows from the duality principle and lemma 1. Let L be a language with subexponential initial index. We construct an oracle O and a machine M with sublinear space bounds, such that L = L(M,O).

Construction of O. For each n, let A_n be a finite automaton defined by

$$A_n = \langle \Sigma , Q_n, \delta_n, 0, F_n \rangle$$

where Q_n can be assumed to be { 0,1,2,... }, such that $L(A_n) = L \cap \Sigma^n$ and $||Q_n|| \in O(2^{o(n)})$ by lemma 18 in the appendix. As a consequence a sublinear number of bits suffices for encoding each state of Q_n. Define the oracle 0 by:

$$0 = \{ n\$x\$i\$j\$u \ / \ j \in \delta_n(i,x) \text{ and } u = \text{if } j \in F_n \text{ then 1 else 0} \}$$

where n, i, and j are written in binary. Observe that the words in 0 have sublinear length, because the code of n is logarithmic and the codes of i and j are sublinear.

Construction of M. Machine M will simulate A_n on inputs of length n. This is done by storing in the work tapes the current state i and the next state j. M obtains j from the oracle by guessing it and querying the oracle about words of the form nxiju. Both the work tapes and the oracle tape are bounded by $\max \{ \log n, \log ||Q_n|| \} \in o(n)$.

Conversely, let L = L(M,0) where M has k work tapes with a sublinear space bound. Consider the automata A_n having states of the form $\langle q, c_0, c_w \rangle$, where q is a state of M, c_0 is a configuration of M's oracle tape, and $c_w = \langle c_1 .. c_k \rangle$ is a configuration of the k work tapes. As M has a sublinear space bound, the number of configurations in each work or oracle tape is in $O(2^{o(n)})$. Hence the number of states of A_n is in $O(2^{o(n)})$. All these facts follow from lemma 18 in the appendix. Transitions among the states are defined by mimicking the transitions of M. It is immediate that $L(A_n)$ is

then $L \cap \Sigma^n$; hence $a_L(n) \in O(2^{o(n)})$. This completes the proof. □

We consider now the on-line versions of the auxiliary pushdown automata defined in [7]. See also [3] and [6]. The conventions regarding space bounds on the work tapes and/or oracle tape are as before. Note that there is no bound on the size of the pushdown store. The theorem we prove amounts, intuitively speaking, to adding a pushdown to "both sides" of the previous theorem.

The corresponding notation for denoting linear lower bounds for this computation model is as follows:

$$\forall\,(\mathrm{Space}(\Omega_\infty(n)) \vee \mathrm{Oracle}(\Omega_\infty(n)), \mathrm{Pushdown})$$

denotes the class of all sets L such that for every auxiliary pushdown machine M and oracle A with L = L(M,A), either the work space or the oracle tape used by M are in $\Omega_\infty(n)$. Dually, we denote

$$\exists\,(\mathrm{Space}(o(n)) \wedge \mathrm{Oracle}(o(n)), \mathrm{Pushdown})$$

the class of all sets L such that there are an auxiliary pushdown machine M and an oracle A with L = L(M,A), such that both the work space and the oracle tape of M are bounded by functions in $o(n)$.

A "duality principle" similar to the stated above holds:

Lemma 4. For any set L,

$L \in \forall\,(\mathrm{Space}(\Omega_\infty(n)) \vee \mathrm{Oracle}(\Omega_\infty(n)), \mathrm{Pushdown})$

iff $L \notin \exists\,(\mathrm{Space}(o(n)) \wedge \mathrm{Oracle}(o(n)), \mathrm{Pushdown})$

Using these classes we obtain a characterization of the languages having exponential context-free cost. We state first some auxiliary definitions and lemmas. From now on, "grammar" is to be understood always as context-free grammar.

Given a grammar G, we define its size as the number of rules it contains, and denote it by $||G||$. For a rule $p : v \longrightarrow w$ of G, its length, denoted by $|p|$, is defined as $|v|+|w|$. The length of a grammar, denoted by $|G|$, is the sum of the lengths of all its rules.

Given a pda P, we define its size as the number of states, and its length as the sum of the lengths of its transitions, where the length of a transition $(q,u,Z) \vdash\!\!\!- (q',Z_1..Z_r)$ is just $|quZq'Z_1..Z_r|$.

The next lemma proves that we can define equivalently context-free cost in terms of lengths of grammars or lengths of pda's.

Lemma 5. For any L, the following are equivalent:

(1) There exists a succession of grammars G_n with $L(G_n) = L \cap \Sigma^n$ such that $||G_n|| \in O(2^{o(n)})$.

(2) There exists a succession of grammars G_n with $L(G_n) = L \cap \Sigma^n$ such that $|G_n| \in O(2^{o(n)})$.

(3) There exists a succession of pda's P_n with $L(P_n) = L \cap \Sigma^n$ such that $|P_n| \in O(2^{o(n)})$.

Proof.

(1) => (2). Transform each G_n into an equivalent G'_n without rules $v \longrightarrow \lambda$. This can be done with small overhead. In the grammar G'_n every rule $v \longrightarrow w$ has length $|v|+|w| \leq n+1$. Hence the length of G'_n is in $n.2^{o(n)} = 2^{o(n)}$.

(2) => (3). Consider a pda having transitions of the form

$$(q, \lambda, v) \vdash (q, w^R)$$

$$(q, x, x) \vdash (q, \lambda)$$

for each rule $v \longrightarrow w$ of the grammar and each terminal symbol x. It has the same size as the grammar.

(3) => (1). As every transition in a pda allows to reach at most one new state, the number of states is also bounded by $2^{o(n)}$. The

classical construction of Ginsburg [13] gives a grammar G_n of size $||G_n|| \in O(2^{\sigma(n)})$.

□

Given a pushdown automaton for $L \cap \Sigma^n$ it is easy to obtain another pushdown automaton such that $Z_n = \{ 0, 1 \}$ where the pushing rules are of the types

(i, u, 0) -> (j, 00)

(i, u, 0) -> (j, 01)

(i, u, 0) -> (j, λ)

(i, u, 1) -> (j, 10)

(i, u, 1) -> (j, 11)

(i, u, 1) -> (j, λ)

This can be done with a small overhead of the size. We call such automata normalized. The relationship between context-free cost and the number of states of a normalized automaton is given by:

Lemma 6. The following conditions are equivalent:

(1) $cf_L(n) \in O(2^{o(n)})$.

(2) There exists a succession of normalized pda P_n, with sets of states Q_n, such that P_n accepts $L \cap \Sigma^n$, and $|Q_n| \in O(2^{o(n)})$.

Note that, by duality, $cf_L(n) \in \omega(2^{\Omega_\infty(n)})$ can be characterized analogously in terms of the number of states of every succession of pda recognizing $L \cap \Sigma^n$.

Now we can state our announced characterization:

Theorem 7. For nondeterministic on-line auxiliary pushdown machines,

(1) $\Omega_\infty(\exp)[cf] = \forall\,(\mathrm{Space}(\Omega_\infty(n)) \vee \mathrm{Oracle}(\Omega_\infty(n)), \mathrm{Pushdown})$

(2) $o(\exp)[cf] = \exists\,(\mathrm{Space}(o(n)) \wedge \mathrm{Oracle}(o(n)), \mathrm{Pushdown})$

Proof.

(1) will follow from (2) by duality. We must prove (2). Let L be in $o(\exp)[cf]$. There exists a succession of normalized automata P_n with sets of states of cardinality $||Q_n|| \in O(2^{o(n)})$. Define the oracle O by:

$$O = \{\, n\$i\$x\$Z_1\$j\$Z_2\$Z_3 \;/\; (i,x,Z_0) \vdash (j,Z_2Z_3) \text{ is a rule of } P_n \,\}$$

Then i and j are always of sublinear length. The simulation by an auxiliary pushdown machine follows the same ideas of theorem 3; the pushdown of P_n is simulated by the auxiliary pushdown.

Reciprocally, assume that L = L(M,O) where M is a sublinear space bounded auxiliary pushdown machine. Without loss of generality, we can assume that M uses its pushdown in the same way as a normalized pda. Consider as states of the pda tuples describing configurations of the work and oracle tapes of M. As the length of each tape is o(n), the number of states is $O(2^{o(n)})$. The transitions of the pda simulate the computation of M over inputs of length n.

□

Our last result of this section relates exponential boolean circuit complexity with a class defined in terms of time and oracle space bounds of Turing machines. Define the following notation:

$$\forall\ (\mathrm{Time}(\Omega_\infty(\exp)) \vee \mathrm{Oracle}(\Omega_\infty(n)))$$

denoting the class of all sets L such that for every oracle Turing machine M and oracle A with L = L(M,A), either the running time is in $\Omega_\infty(\exp)$ or the amount of oracle tape used by M is in $\Omega_\infty(n)$. Dually, we denote

$$\exists\ (\mathrm{Time}(o(\exp)) \wedge \mathrm{Oracle}(o(n)))$$

the class of all sets L such that there are an oracle Turing machine M and an oracle A with L = L(M,A), such that the running time of M is bounded by a function in o(exp) and the oracle tape of M is bounded by a function in o(n). Again we can state a "duality principle":

Lemma 8. For any set L,

$$L \in \forall\ (\mathrm{Time}(\Omega_\infty(\exp)) \vee \mathrm{Oracle}(\Omega_\infty(n)))$$
$$\text{iff } L \notin \exists\ (\mathrm{Time}(o(\exp)) \wedge \mathrm{Oracle}(o(n)))$$

Using these classes we prove the following:

Theorem 9. For deterministic Turing machines,

(1) $\Omega_\infty(\text{exp})[c] = \forall\,(\text{Time}(\Omega_\infty(\text{exp})) \vee \text{Oracle}(\Omega_\infty(n)))$

(2) $o(\text{exp})[c] = \exists\,(\text{Time}(o(\text{exp})) \wedge \text{Oracle}(o(n)))$

Proof.

Again we only prove part (2); (1) will follow by duality. Let L be a language with subexponential cost. We construct a machine M and an oracle O such that M accepts L in subexponential time using sublinear oracle tape.

Construction of O. Consider the circuit corresponding to inputs of size n. Each gate can be numbered by counting up to some value f(n) $\in O(2^{o(n)})$. The oracle contains information about the value f(n), and also information about the connections of the circuit. The value of f(n) is needed by the deterministic machine in order to know when to stop querying the oracle for new gates. Define:

$$O = \{\, n\$1^{f(n)} \,\} \cup$$

$$\{\, n\$\$i\$j\$k\$\theta \;/\; \text{gate } i \text{ of the circuit for length } n \text{ is the operation } \theta \text{ over gates } j,k \,\}$$

Observe that the words in O needed to find the circuit for inputs of length n are of length o(n), because i, j, and k can be written down with a sublinear quantity of bits.

Construction of M. The machine M works in two different phases. First, it constructs the circuit corresponding to the length n of the input, with the help of the oracle 0; after, it evaluates the circuit over the input. As the number of gates is subexponential, the whole circuit can be constructed and evaluated by levels within subexponential time. We omit this second procedure, which is well-known.

Let us present the construction of the circuit. First, the machine queries the oracle successively about the words of the form $n\$1^t$ for t = 1,2..., storing the current value of t in a separate tape. A positive answer means that the tape contains the maximum number of a gate of the circuit, f(n). Then the machine queries 0 about all the words of the form $n\$i\$j\$k\$\theta$ for i, j, and k less than or equal to f(n), and each allowed binary boolean operation θ. Each time a positive answer is received, it means that one more gate is known and stored in the circuit tape. As f(n) is in o(n), the time employed in the construction is $0(2^{o(n)})$.

This machine accepts L in subexponential time with oracle 0.

Conversely, let L = L(M,0) where M works in subexponential time and uses oracle space f(n), where f is a sublinear function. Let

$$c_0 : \{ 0, 1 \}^{f(n)} \longrightarrow \{ 0, 1 \}$$

be the characteristic function of 0 up to size f(n). The representation of c_0 as a sum of minterms requires $o(n).2^{o(n)}$ "and"

gates and $2^{o(n)}$ "or" gates, which is a circuit of subexponential size.

Now a circuit can be built for the machine M assuming no oracle queries are made, as in [16]. The size of the circuit is the square of the running time of M: this is

$$(2^{o(n)})^2 = 2^{o(n)+o(n)} = 2^{o(n)}$$

and hence is also subexponential.

Finally, combine both circuits by plugging a copy of the circuit for the oracle into each step of the machine's computation, so that if on some input a query is made at this step of the computation, it can be correctly answered. The full circuit is again subexponential by the same argument on the square as above, and it is able to decide L for inputs of length n. Therefore the boolean cost of L is subexponential, as was to be shown.

□

IV. Upper bounds.

Given a nonuniform measure we can define families of languages by imposing upper bounds on the measure. In this section three classes of functions are considered as upper bounds: logarithmic functions, polylog (i.e. $(\log n)^k$) functions, and polynomials. For any nonuniform measure m, the family of languages defined by polynomial upper bounds is:

$$O(poly)[m] = \{ L \;/\; \exists\, k \in N \text{ with } m_L(n) = O(n^k) \}$$

When m is the initial index, the context-free cost, or the boolean size complexity, the families defined in this way were characterized in [1] (the last case is due to A. Meyer). These characterizations used the notion of "sparse oracles", with no bound set over the length of the oracle tape. Define the class SP of all the so-called sparse oracles, as the class formed by all the oracle sets consisting of a polynomially growing amount of words. The characterizations presented there were the following:

Theorem 10. [1]

(1) Consider nondeterministic log-space on-line oracle Turing machines; denote $NLOG_{on}(S)$ the corresponding uniform class for oracle S. Then

$$O(poly)[a] = \bigcup_{S \in SP} NLOG_{on}(S).$$

(2) Denote $ANLOG_{on}(S)$ the analogous class defined by on-line log-space auxiliary pushdown machines. Then

$$O(poly)[cf] = \bigcup_{S \in SP} ANLOG_{on}(S).$$

(3) Let P(S) denote the standard polynomial time class relativized

to S. Then

$$O(poly)[c] = \bigcup_{S \in SP} P(S).$$

Several corollaries were drawn regarding both uniform and non-uniform complexity classes, and similar characterizations were shown for other nonuniform measures.

We present here another set of characterizations of classes defined by polynomial upper bounds, based on the space available for oracle queries, rather than on the structure of the oracle set. We characterize also the dual family

$$\omega(poly)[m] = \{ L \;/\; m_L(n) \in \omega(poly) \}$$

for which a duality principle with O(poly)[m] can be stated.

For the remaining of this section, our log-space machines have also a logarithmic bound on the length of the oracle tape.

Theorem 11. Consider on-line oracle Turing machines and on-line auxiliary pushdown machines as in the previous section. The classes that appear in the following are defined exactly as before.

(1) $O(poly)[a] = \exists (Space(O(\log n)) \wedge Oracle(O(\log n)))$.

(2) $\omega(\text{poly})[a] = \forall\,(\text{Space}\,(\omega(\log n)) \vee \text{Oracle}\,(\omega(\log n)))$.

(3) $O(\text{poly})[cf] = \exists(\text{Space}(O(\log n)) \wedge \text{Oracle}(O(\log n)),\text{Pushdown})$

(4) $\omega(\text{poly})[cf] = \forall(\text{Space}(\omega(\log n)) \vee \text{Oracle}(\omega(\log n)),\text{Pushdown})$.

(5) $O(\text{poly})[c] = \exists\,(\text{Time}\,(O(\text{poly})) \wedge \text{Oracle}\,(O(\log n)))$

(6) $\omega(\text{poly})[cf] = \forall\,(\text{Time}\,(\omega(\text{poly})) \vee \text{Oracle}\,(\omega(\log n)))$.

The even-numbered statements follow from the odd-numbered statements by duality principles. Proofs of the odd-numbered statements can be constructed by rephrasing the proofs of theorems 3, 7, and 9, adjusting the bounds to polynomials and checking that the necessary conditions of closure of the class of bounding functions hold for polynomials.

Observe that (5) (and similarly (1) and (3)) can be interpreted in the following manner: a set L has polynomial size boolean circuits iff there is an (arbitrary) oracle set A such that L is in P(A), and the polynomial time machine witnessing this fact uses only a part of the oracle tape which is bounded by a logarithm. About the family $\omega(\text{poly})[a]$, it is possible to define quite natural sets L (see [5]) such that $a_L(n) \in \theta(n^{c \cdot \log n})$, and hence $L \in \omega(\text{poly})[a]$.

In [1] only polynomials were used as upper bounds for defining non-uniform classes. For this bound, the exact relationship between boolean formulae (or, equivalently, circuits of fan-out 1) and branching programs is open, although it is generally conjectured that they are not equivalent. A branching program is a directed, acyclic graph with an initial and some final nodes. Nodes are labeled by integers. Each internal node has exactly two outcoming edges labeled 0 and 1. Its computation starts at the initial node, and at node with label n it follows the path labeled with the value of the n^{th} bit of the input. See [21], for example, for material related to branching programs. However, the following can be proven:

Theorem 12. A set L has polynomial size branching programs iff there is an (arbitrary) oracle set A such that L is in $DLOG_{off}(A)$.

Proof.

Let L have polynomial size branching programs. Construct an oracle having words of the form nijuk, meaning that node of code i in branching program n is labeled by j, and the outcoming edge from i labeled by u (which is 0 or 1) goes to node k. Then a logspace machine can simulate it by keeping the current node i and the position of the input head in the work tapes, cycling over all the words of the form nij0k and nij1k until the oracle indicates the correct one, bringing the input head to position j on the input tape, and deciding the correct path. If no such word is in the oracle then a final node has been reached. It is easy to see

that the required space is logarithmic.

Conversely, each configuration of the machine, including the oracle tape, is considered to be a node of the branching program. Its label is the position of the input head in the configuration, and the edges labeled 0 and 1 are defined by the transitions of the machine and the answers of the oracle. The size of such branching program is polynomially bounded.

□

Let us turn for a moment to the depth of the circuits for a set. Polynomial bounds no longer have a meaning, because every boolean function can be synthesized with circuits of linear depth [17]. In the following we consider polylog functions and we use them to bound the depth of the circuits. If only depth is bounded and no bound is set over the size, then there is no difference between unbounded fan-out and fan-out 1, because fan-out may be trivially reduced to 1 "unfolding" the circuit, with no increment of the depth. Denote as d_L the nonuniform measure which gives the minimum depth needed to synthesize the characteristic function of $L \bigcap \{ 0, 1 \}^n$.

We shall use the following well-known results about depth and size of formulae: first, $d_L(n) \in \theta (\log c_L(n))$ (Spira's theorem; see [17]); and second, if L is in DSPACE(S) then $d_L(n) \in O(S^2(n))$ [2].

Recall the standard notation of [14] for nonuniform complexity

classes: a set A is in the class C/F iff there is a set B in C and a function h from $\{0\}^*$ to Σ^* whose length is bounded by a function of F, such that

$$x \in A \text{ iff } \langle x, h(0^{|x|}) \rangle \in B.$$

A very interesting situation arises when considering polylog functions and n^{polylog} functions as upper bounds. We summarize our results (which we found quite unexpected) in the following theorem. In it, Turing machines are always off-line deterministic machines.

Theorem 13. The following classes are the same:

(1) O(polylog)[d].

(2) The class of sets having boolean formulae of n^{polylog} size.

(3) $\exists$ (Space (O(polylog)) $\wedge$ (Oracle (O(polylog))).

(4)

$$\bigcup_{S \in SP} \text{SPACE}(\text{polylog}, S)$$

when the Turing machines use polylog work space but there is no upper bound on the length of the oracle tape.

(5) SPACE(polylog)/$n^{polylog}$.

(6) The class of sets having its initial index, measured by 2-way finite automata, bounded above by a $n^{polylog}$ function.

(7) The class of sets having branching programs of $n^{polylog}$ size.

Proof.

(1) <=> (2). Immediate from Spira's theorem.

(1) <=> (3). Let L have formulae of polylog depth. The size of these formulae cannot be more than $2^{polylog} = n^{polylog}$ (see lemma 19 in the appendix). Assigning a number to each node in the formula for length n, we can encode the formulae into an oracle 0 by words of the form nijkθ exactly like in theorem 9. We evaluate the formula using top-down computation. A stack of polylog size suffices, because both the depth of the formula and the size of each node are bounded by a polylog function.

Conversely, let L = L(M,0) with polylog space. The oracle can be encoded into a polylog depth formula as in theorem 9. The simulation of the polylog-space machine M, without queries, can be done in polylog depth by the result of Borodin [2] cited above.

(1) <=> (4). The proof is very similar to (1) <=> (3). The only

change is that the oracle is formed by words of the form $0^n\$i\$j\$k\$\theta$. It is easy to show that such oracle must be sparse. Of course the bound on the length of the oracle tape no longer holds; this is the only change on the accepting machine. For the converse, it is very easy to see that sparse oracles can be encoded in polynomial depth formulae. The results in [1] follow the same technique.

(2) => (5). Consider an advice function such that for each n it gives the formula corresponding to length n. It is bounded in length by a polylog function. It can be evaluated within polylog space as in the case (1) <=> (3).

(5) => (3). Given the advice function h of polylog length for L, we construct an oracle 0 with words of the form n$x where x is a prefix of h(n). A polylog space machine M can obtain this advice from the oracle and use it to decide L. The oracle space is also polylog.

(3) <=> (6). It is exactly like (1) in theorem 11, with the only change that both the automata and the polylog space machines have two-way input heads.

(6) <=> (7). As theorem 12 for polylog bounding functions.

□

V. Some trivial families.

We have analyzed classes defined either by imposing high lower

bounds or by imposing low upper bounds on the nonuniform complexity of the sets. Let us make some remarks on the "trivial" families defined by high upper bounds and by low lower bounds.

For the former, it is very easy to prove that every set can be recognized by a family of automata of exponential size. Hence, both classes O(exp)[a] and O(exp)[c] contain every set.

For the latter, our motivation is the problem of finding a characterization of the class

$\exists$ (Space (o(log n)) $\wedge$ Oracle (o(log n)))

First observe that Ω_∞(poly) = Ω_∞(n). In [9] it is shown that the initial index of L is o(n) iff L is finite. We can state the following:

Theorem 14. The following are equivalent:

(1) L is in o(n)[a].

(2) L is in o(poly)[a].

(3) L is finite.

(4) L is in $\exists$ (Space (o(log n)) $\wedge$ Oracle (o(log n)))

Proof.

The equivalence of (1) and (2) is immediate. The equivalence of (1) and (3) was established in [9], and is not difficult to prove. Any finite set L can be decided with no oracle nor work tape space by a Turing machine which simulates in its finite control a finite automaton for L. Finally, if L fulfills (4) then its initial index is sublinear, by the same argument used in theorem 3: the number of configurations of such a machine is bounded by $2^{o(\log n)} = o(n)$.

□

By duality, a set L is infinite iff its initial index is in $\Omega_\infty(\text{poly})[a]$, and iff it is in the class

$\forall$ (Space($\Omega_\infty(\log n)$) $\vee$ Oracle($\Omega_\infty(\log n)$).

A similar reasoning can be presented about combinational complexity. Recall from [17] the definition of a boolean function being essentially dependent of $\Omega_\infty(n)$ variables and the following property: $c_L(n)$ is $\Omega_\infty(n)$ iff the characteristic function of L is essentially dependent of $\Omega_\infty(n)$ variables. Then a proof similar to the proof of theorem 14 leads to:

Theorem 15. The following are equivalent:

(1) L is in $o(n)[c]$.

(2) The characteristic function of L depends essentially of $o(n)$

variables.

(3) L is in $\exists$ (Time (o(n)) $\wedge$ Oracle (o(log n)))

VI. Conclusions.

Our main results are characterizations of classes defined by upper and lower bounds on some nonuniform complexity measure. We characterize these classes by considering standard computational devices (oracle Turing machines) and bounding simultaneously some resources like the work space or the running time, and the length of the queries. The first bound is a bound on the computational capabilities of the machine, whereas the second bound gives an estimate of the amount of external information given to the machine.

Previous results [1] gave other characterizations depending on the structure of the oracle set. These results could suggest a trade-off between the amount of work space or time and the amount of information coded by the oracle, similar to the trade-offs known for other computational resources (like time for space in [12]). However, our characterizations (for arbitrary oracles) in terms of the way the machines query the oracle show that this trade-off only holds in a very trivial way, where the two extreme situations are "almost everything is given by the oracle" and "almost everything is computed", and NO intermediate nontrivial situation holds. Let us present a simple example of this fact. Consider the language

$C = \{ w\$w \mid w \in \{0,1\}^* \}$ of the squares with a central separator. It is not difficult to show [9] that its initial index is exponential. Our first theorem shows that in order to recognize this set we need a linear amount of space. On the one side, a machine may use no oracle space, copy w in a separate tape until the $ is reached, and then check the second half; we need linear work space. On the other side, a machine can use no work space, just copy the input in the oracle tape, and then query an oracle for C; we need linear oracle space. By our first theorem, we know that we can do no better: any machine for C must use either linear work space or linear oracle space, and hence the two trivial machines just presented are in some sense optimal. A similar argument for auxiliary pushdown machines follows from our theorems and the fact that C has exponential context-free cost [10]. We find this result quite surprising.

Furthermore, we have shown that this technique works for already well-studied classes as the ones defined by polynomial upper bounds. A duality principle can be always stated relating an upper bound with its corresponding lower bound, and this duality carries over to the uniform characterizations, giving the lemmas stated here as "duality lemmas". This phenomenon provides new characterizations of classes defined by polynomial and $n^{polylog}$ upper bounds. Finally, we have shown that measures that we expected to differ for polynomial upper bounds, like formula size and branching program size, do not differ for these $n^{polylog}$ upper bounds. For this family we have obtained several different characterizations, some of them quite surprising. We consider that this collection of results presents an

interesting new view of the landscape of nonuniform complexity classes.

VII. Appendix.

We develop here some properties of orders of magnitude. A fundamental reference about orders of magnitude is [15]. However, we follow [20] in our definitions. Given a function f, we denote:

(1) O(f) is the set of functions g such that for some $c > 0$ and for every but finitely many n, $g(n) < c.f(n)$.

(2) o(f) is the set of functions such that

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

(3) $\Omega_\infty(f)$ is the set of functions g such that for some $c > 0$ and for infinitely many n, $g(n) > c.f(n)$.

(4) $\omega(f)$ is the set of functions g such that

$$\limsup_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

Let U be the set of all functions from N to N. The following fact holds:

Lemma 16. For every $g \in U$, $\Omega_\infty(g) \cup o(g) = U$, and $\Omega_\infty(g) \cap o(g) = \emptyset$.

Consider exponential lower bounds. We denote the family of functions with an exponential lower bound (exponentials for short) by

$$\Omega_\infty(\exp) = \bigcup_{c>0} \Omega_\infty(2^{cn}).$$

Similarly the family of subexponentials is

$$o(\exp) = \bigcap_{c>0} o(2^{cn}).$$

For these families the same property as above holds:

Lemma 17. $\Omega_\infty(\exp) \cup o(\exp) = U$, and $\Omega_\infty(\exp) \cap o(\exp) = \emptyset$.

A more interesting relation holds between the linear and the exponential functions:

Lemma 18. $o(\exp) = O(2^{o(n)})$.

Proof.

Let f be in $\bigcap_{c>0} o(2^{cn})$. Define the function g as $g(n) = \lceil \log_2 f(n) \rceil$. For every $c > 0$ we have that

$$\lim_{n \to \infty} 2^{\lceil \log f(n) \rceil - cn} = 0$$

As n grows, the difference $\log f(n) - cn$ goes to $-\infty$, and then

$$\lim_{n \to \infty} \frac{\log f(n)}{n} = 0$$

which implies that $g(n)$ is in $o(n)$. As f is in $O(g)$, the inclusion from right to left is proven.

For the converse, let $g(n)$ be a function in $o(n)$. It suffices to prove that $2^{g(n)}$ is in $o(2^{cn})$ for each $c > 0$. We know that

$$\lim_{n \to \infty} \frac{g(n)}{n} = 0$$

and this implies that for each c, and all but finitely many n, $g(n) < (c/2)n$. Hence $g(n) - cn < (-c/2)n$, which tends to $-\infty$. Thus for every $c > 0$

$$\lim_{n \to \infty} g(n) - cn = -\infty$$

and therefore

$$\lim_{n \to \infty} \frac{2^{g(n)}}{2^{cn}} = 0$$

which was to be shown. □

The intuitive interpretation of this lemma is clear: it means just that a function is subexponential iff it can be written down in a sublinear number of bits.

Some results are also needed about polynomial and polylog upper bounds. Define:

$$O(\text{poly}) = \bigcup_{k>0} O(n^k)$$

$$O(\text{polylog}) = \bigcup_{k>0} O((\log n)^k)$$

$$O(n^{\text{polylog}}) = \bigcup_{k>0} O(n^{(\log n)^k})$$

We use the following property of this last classes:

Lemma 19. $O(n^{\text{polylog}}) = O(n^{O(\text{polylog})}) = O(2^{O(\text{polylog})})$.

Here the first equality is proven similarly to lemma 18. The equality of the two last classes follows from the fact that $n^{(\log n)^k}$ coincides with $2^{(\log n)^{k+1}}$. We leave the proof to the

reader.

We close this appendix with some observations regarding the classes $\theta(f)$ as defined in [15]. It represents the functions within a "band" around f. We denote it θ_K. It allows to strengthen the lemmas proven here, yielding the following:

Lemma 20.

(1) $o(exp) = \theta_K(2^{o(n)})$

(2) $\Omega_\infty(exp) = \theta_K(2^{\Omega_\infty(n)})$

(3) $O(poly) = \theta_K(2^{O(\log n)})$

(4) $\omega(poly) = \theta_K(2^{\omega(\log n)})$

This leads to a result, dual of lemma 18, whose meaning is that a function has an exponential lower bound iff it needs at least a linear number of bits to be written down. On the other hand, it is interesting to note that the highly different approaches for defining lower bounds presented in [15] and [20] interact fairly well, as this last lemma shows.

References.

[1] J.L. Balcázar, J. Díaz, J. Gabarró: Uniform characterizations of nonuniform complexity measures. To appear in Information and Control.

[2] A. Borodin: On relating time and space to size and depth. SIAM J. Comp. 6, 4 (1977), 733-744.

[3] F. Brandenburg: On one-way auxiliary pushdown automata, 3rd GI conf. on Theor. Comp. Sci. (1977), Springer Verlag, LNCS 48, 132-144.

[4] W. Bucher, K. Culik, H. Maurer, D. Wotschke: Concise description of finite languages. Theor. Comp. Sci. 14, 3 (1981), 227-246.

[5] R. Casas, J. Gabarró: About LOG-ON languages. Internal report RR 85/02, Facultat d'Informática de Barcelona.

[6] M. Chytil: Almost context-free languages. Manuscript (1984).

[7] S. Cook: Characterizations of pushdown machines in terms of time-bounded computers. Journal ACM 18, 1 (1971), 4-18.

[8] A. Ehrenfeutch, G. Rozenberg: On the separating power of EOL systems, RAIRO Inf. Theor. 17, 1 (1983), 13-22.

[9] J. Gabarró: Funciones de complejidad y su relación con las familias abstractas de lenguajes. Ph. D. dissertation, 1983. See also: Initial index: a new complexity function for languages, in ICALP 83, Lect. Notes in Comp. Sci. 154, 226-236.

[10] G. Goodrich, R. Ladner, M. Fischer: Straight-line programs to compute finite languages. Conf. Theor. Comp. Sci., Waterloo 1977.

[11] M. Harrison: Introduction to switching and automata theory, McGraw Hill, New York 1965.

[12] J. Hopcroft, W. Paul, L. Valiant: On time versus space and related problems. Journal ACM 2 (1977), 332-337.

[13] J. Hopcroft, J. Ullman: Introduction to automata theory, languages, and computation, Addison-Wesley, Reading (Mass.) 1979.

[14] R. Karp, R. Lipton: Some connections between nonuniform and uniform complexity classes, 12 ACM Symp. Th. of Comp. (1980), 302-309.

[15] D. Knuth: Big omicron and big omega and big theta. SIGACT News Apr-June 1976, 18-24.

[16] R. Ladner: The circuit value problem is log space complete for

P. SIGACT News Enero 1975, pp 18-20.

[17] J. Savage: The complexity of computing, Wiley Interscience, 1976.

[18] C. Schnorr: The network complexity and the Turing machine complexity of finite functions. Acta Informática 7 (1976), 95-107.

[19] M.J. Serna: in preparation.

[20] P. Vitanyi, L. Meertens: Big omega versus the wild functions. EATCS Bulletin 22, Feb. 1984, 14-19.

[21] I. Wegener: On the complexity of branching programs and decision trees for clique functions. Preprint, 1984.