ON SOME "NON-UNIFORM" COMPLEXITY MEASURES

José Luis BALCÁZAR
Josep DÍAZ
Joaquim GABARRÓ

RR84/05

ON SOME "NON-UNIFORM" COMPLEXITY MEASURES

J. L. Balcázar, J. Díaz, J. Gabarró
Facultat d'Informática de Barcelona
Universitat Politécnica de Catalunya, 08034 Barcelona

Abstract:

Non-uniform complexity measures origined in Automata and Formal Languages Theory are characterized in terms of well-known uniform complexity classes. The initial index of languages is introduced by means of several computational models. It is shown to be closely related to context-free cost, boolean circuits, straight line programs, and Turing machines with sparse oracles and time or space bounds.

Resum:

Es caracteritzen mesures no uniformes de complexitat originades en Teoria d'Autòmats y Llenguatges Formals, mitjançant classes uniformes. Es presenten definicions de l'índex inicial dels llenguatges formals amb diversos models de computació. Es demostra la relació d'aquests amb el cost incontextual, els circuits booleans, els programes rígids i les màquines de Turing amb oracles esparsos i fites de temps o d'espai.

I. Introduction.

The study of the complexity of formal languages has been done last times from several different viewpoints. Our work is centered on two of the most succesful of them: on the one hand, bounds on concrete resources used by algorithms in deciding the languages, such as time and memory; on the other one, functions describing the growth of descriptions of the finite initial subsets of the language. We call these two approaches, respectively, "uniform complexity" and "non-uniform complexity", following an usual practice. Names seem to be origined by the fact that the first approach studies the complexity of algorithms deciding the whole language in a uniform manner, while the second allows a different description for each initial subset without asking all of them to perform similar computations.

The uniform measures we shall use rely on the multitape Turing machine (TM for short) as a formal model of algorithm. This model is described later. Time is identified with number of elementary steps in the computation; memory is identified with space used in the work tapes. Bounds on these resources will define our uniform complexity classes. Sometimes our model will have restrictions in the moves of the input tape head: we consider "on-line" machines, whose input can be read only once. Nondeterministic machines will also be used.

Some non-uniform measures are classical notions of the Theory of Computing; one of the most widely used ones is the circuit size measure (see, v. gr., [Sa,76]). It is defined by counting the number of boolean gates needed to sythesize the characteristic funtion of a finite set. Other non-uniform measures have originated from Automata and Language Theory, like context-free cost [BCMW,81],

or initial index [Ga,83]. Also, straight line programs with set-theoretic operations, as in [GLF,77] will be considered.

This work is based on a characterization usually credited to A. Meyer. He used oracles in order to "break up" the uniformness of the TM model, proving that a set can be described by a polynomially growing set of boolean circuits if and only if it was possible to recognize this set by a TM within polynomial time, with the aid of a "sparse" oracle. Sparse oracles have only a polynomially growing number of words. We use similar techniques to characterize polynomial classes under several non-uniform complexity measures. We prove, for example, that a set can be described by a polynomially growing set of finite automata if and only if it is possible to recognize this set by an on-line TM within log-space with the aid of a sparse oracle. Similarly, a set can be described by a polynomially growing set of context-free grammars if and only if it is possible to recognize this set by an on-line TM within logarithmic space, with the aid of a sparse oracle and of an unbounded pushdown store. Other characterizations are proven along the way.

In particular, we analyze the initial index, measured with finite automata, in section II. We characterize the polynomial class for this measure and we obtain several interesting consequences about the uniform class $NLOG_{on}$: this class is not closed under complements, and possess complete sets with respect to on-line log-space reductions. We study in section III the context-free cost, and we characterize it in terms of straight line programs, initial index measured by pushdown automata, and on-line versions of the auxiliary pushdown machines of Cook [Co,71]. Complete sets are found for some of the uniform classes, as well as properties like non-closure under complements; it is interesting to note that the off-line classes studied by Cook are however closed

under complements. Section IV is devoted to the study of circuit size complexity, which we characterize by means of more powerful versions of straight line programs. In a short final section we revise the results of the previous ones, and suggest some lines of further research and a conjecture about formula size and off-line log-space.

On-line TM's are, to our opinion, an interesting model which should be studied more deeply; when the memory bounds are less than linear, we consider that off-line machines are somewhat irrealistic, because access to the whole input file in both directions is not usually feasible in the present computers. Sequential files do not usually allow to go back to the previous record, while random access files should be considered as work space and taken in account when measuring the space complexity of the algorithms that use them.

We consider that the facts proven in this paper, and the techniques used in the proofs, support as an intuitive consequence that the borderline between Automata and Formal Language Theory and Complexity Theory lies in some sense nearby the question of whether the computational model chosen has or has not the ability of reading back its input. On-line models are more adequate when dealing with notions arisen from Automata Theory than off-line models are.

II. Rational complexity.

The first non-uniform measure which we consider is the initial index, due to Gabarró [Ga,83].

Definition 1. Given a language $L \subset \Sigma^*$, we define the initial index of L as the function $a_L : N \dashrightarrow N$ given by:

$$a_L(n) = \min \{ ||A|| \ / \ A \text{ is a nondeterministic automaton such that } L(A) = L \cap \Sigma^n \}$$

where $||A||$ denotes the number of states in A. In the same manner we define the deterministic initial index da_L as:

$$da_L(n) = \min \{ ||A|| \ / \ A \text{ is a deterministic automaton such that } L(A) = L \cap \Sigma^n \}.$$

From these two complexity measures we define the following complexity classes:

$$Pol_a = \{ L \ / \ \exists\, k \in N \text{ with } a_L(n) = O(n^k) \}$$
$$Pol_{da} = \{ L \ / \ \exists\, k \in N \text{ with } da_L(n) = O(n^k) \}.$$

It is easy to show that if L_p is the set of palindromes over a two letter alphabet, then its complement $\bar{L}_p$ has the property

$$\bar{L}_p \in Pol_a \text{ but } \bar{L}_p \notin Pol_{da}$$

(see [Ga,83]) and therefore $Pol_{da} \subsetneq Pol_a$.

We consider as our first uniform complexity model the on-line oracle Turing machines working within space logarithmic in the length of the input. On-line machines have been considered previously in the literature [HLS,65], [HU,69].

An on-line oracle machine M is a multitape Turing machine, deterministic or nondeterministic, with a read-only input tape; k read-write work tapes; a distinguished write-only tape called the query tape; and three distinguished states called QUERY, YES, and NO. The input head moves left to right, and it cannot back to the left. At some moments in the computation, M can write symbols on the query tape; when M enters the QUERY state, it transfers to the state YES if the contents of the query tape is in some oracle set B; otherwise, M transfers into the state NO. In either case the query tape is instantly erased.

As usual, Turing machines are described as tuples $\langle Q, \Sigma, k, \delta, q_0, F \rangle$ where Q is the set of states, Σ is the alphabet, k is the number of tapes, δ is the transition function, q_0 is the initial state, and F is the set of accepting states. The language accepted by such a machine M relative to an oracle B, denoted L(M,B), is defined also in the usual way.

A machine M may be forced to operate within log-space. The machine is started in an initial configuration in which the work tapes have begin and end markers, leaving in between only a logarithm of the length of the input as work space. If attempt is made to cross left of the begin marker or right of the end marker, the computation is aborted and the machine stops in a rejecting state. Observe that no bound is set over the length of the oracle tape.

By a work tape configuration of a machine we mean a description of the contents of each work tape, including information about the current position of the work tape head.

This model of computation defines the following complexity classes:

$NLOG_{on}(B)$ = { L / L is accepted by a nondeterministic on-line machine with oracle B within log-space }

$DLOG_{on}(B)$ = { L / L is accepted by a deterministic on-line machine with oracle B within log-space }

In order to compare uniform an non-uniform measures we will "break up" the uniformity with the aid of sparse oracles. A set S is sparse if there is a polynomial p(.) such that for every n it holds that

$$||\Sigma^n \cap S|| \leq p(n).$$

Given an alphabet Σ, SP will denote the class of all sparse subsets of Σ^*.

Theorem 1.

(i) $Pol_a = \bigcup_{S \in SP} NLOG_{on}(S)$.

(ii) $Pol_{da} = \bigcup_{S \in SP} DLOG_{on}(S)$.

Proof. We shall show first that if $L = L(M,S)$ for some nondeterministic on-line machine M within log-space with a sparse oracle S, then L has polynomial initial index.

For any input w with $|w| = n$, the log-space bound implies a polynomial bound on the number of possible work tape configurations of M. Hence, for each n we can construct from M a new nondeterministic machine M_n accepting $L \cap \Sigma^n$ which uses the same oracle S and no work space, by incorporating into the finite control of M_n all the work tape configurations of M on words of length n. The number of states of M_n is bounded by a polynomial in n.

The working time of M_n being bounded by a polynomial, only a polynomial length of words in the oracle can be queried by M_n to S, which implies by the sparseness of S that only a polynomial number of words can be queried by M with positive answer. We shall construct a new machine M_n' which incorporates also in its finite control a finite automaton A_n for the accesible part of S. These automata A_n can be trivially constructed having size polynomially bounded on n.

Thus, the states of M_n' will be of the form $\langle q, r, C \rangle$, where q is a state of M, r is a state of A_n, and C is a work tape configuration of M. For any symbol x currently scanned by the input head, the transitions of M_n' are of the form:

$$(\langle q, r, C \rangle, x) \vdash (\langle q', r', C' \rangle, k)$$

where $k = 1$ if the input head moves and $k = 0$ if it does not move. The state $\langle q', r', C' \rangle$ is defined as follows:

(i) If q is not the QUERY state and M does not write on the query tape when in configuration $\langle q, C \rangle$, then $r = r'$ and q' and C' are such that $\langle q, C \rangle \vdash \langle q', C' \rangle$ is a transition of M;

(ii) if M writes symbol y on the query tape, then C = C', q' is given by the transition function of M, and r' is given by the transition function of A_n applied to (r, y);

(iii) if q is the query state, then q' is the YES state if r is a final state of A_n, and q' is the NO state otherwise; moreover r' is the initial state of A_n, and C' = C.

This machine is a finite automaton with λ-transitions and size polynomial in n. The elimination of the λ-transitions does not increase the number of states by more than a polynomial [HU,79].

On the contrary, let $L \in Pol_a$. Then there exists a family $\{ A_n / n \geq 0 \}$ of nondeterministic finite automata such that $L(A_n) = L \cap \Sigma^n$. We construct an on-line nondeterministic machine with a sparse oracle S which accepts L within logarithmic space.

Let $A_n = \langle \Sigma, Q_n, \delta_n, 0, F_n \rangle$, where Q_n is assumed to be a sequence of nonnegative integers. Let \$ be a new symbol. We codify the family of automata in the oracle S by defining:

$$S = \{ 0^n\$x\$i\$j\$u \;/\; j \in \delta_n(i,x), \text{ and } u = 1 \text{ if } j \in F_n \text{ and } u = 0 \text{ otherwise} \}.$$

As the size of the automata grows polynomially, it is easy to see that S is sparse. Let p(.) be a polynomial bounding the number of words in S and let b be an integer such that p(n) can be written in log n symbols in base b.

The machine M will have three tapes, named 1, 2, and 3, which at each time will held representations of, respectively, the current state of A_n, the next state of A_n and the length of the input (in binary). We shall denote by i and j the contents of tapes 1 and 2 respectively. Let x denote the symbol currently scanned by the input head. The following nondeterministic procedure accepts input w, with |w| = n, iff $w \in L(A_n)$, and M can be programmed according to it.

```
Begin
x := first symbol of input;
write 0 on tape 1;                          -- initial state
while input last loop
    guess j on tape 2;                      -- next state
    write n-1 on tape 3;                    -- by filling it of ones
    guess whether u = 0 or u = 1;           -- in the finite control
    write on the query tape 0^n$x$i$j$u;
                                            -- use tape 3 to write 0^n
                                            -- copy i from tape 1
                                            -- copy j from tape 2
    query the oracle about this word;
    if the answer is YES then
        copy contents of tape 2 on tape 1;
        x := next symbol of input
    else reject                             -- wrong guess
end while
if u = 1 then accept
else reject
end.
```

We have $w \in L(M,S)$ iff $w \in L(A_n)$. This shows that $L \in NLOG_{on}(S)$.

The proof of the second part of the theorem is similar. The only difference relies in that the procedure describing the actions of the log-space machine which simulates the finite automata must no longer be nondeterministic, and a systematic search over all possible new states j is substituted for the guesses. Observe that in the nondeterministic case no such systematic search could substitute the guesses, because if a wrong computation of the simulated nondeterministic automaton is taken there is no possibility of backtracking the input head.

□

As a corollary of theorem 1, we can state the following known result [HU,69], [Gr,76]:

Corollary 1. $DLOG_{on} \subsetneq NLOG_{on}$.

Proof. The set of the palindromes is not in $NLOG_{on}$, because its initial index is exponential [Ga,83]. However, is complement is; hence $NLOG_{on}$ is not closed under complements.

□

We can use the ideas from the construction in theorem 1 to establish a completeness result for the class $NLOG_{on}$ with respect to reductions computed by deterministic on-line log-space transducers. let us consider the following problem, which we could call the "Replicated Automaton Problem" ("RAP" for short). For any automaton A and any input word $w = x_1x_2...x_n$, RAP has as input $A\$x_1\$A\$x_2\$A\$...\$A\$x_n\$A$, and the problem consists of deciding whether w is accepted by A. Therefore we can specify:

$$RAP = \{ A\$x_1\$A\$x_2\$A\$...\$A\$x_n\$A \;/\; w = x_1x_2...x_n, \text{ and } w \in L(A) \}.$$

It is easy to see that RAP is in $NLOG_{on}$: apply the following procedure, which works on-line in log-space:

```
Begin
read symbols coding A and identify its initial state;
q := initial state of A;
read until $;
while input last loop
    read x_i;
    read symbols coding A and identify q' = δ(q,x_i);
    q := q'
end while;
if q is a final state then accept
else reject
end.
```

To prove the completeness of RAP, we shall build for any log-space on-line nondeterministic Turing machine M a function f, computable by a deterministic on-line log-space transducer which constructs for each w a specific instance of RAP in such a way that $w \in L(M) <=> f(w) \in RAP$.

The computation of f is given by the following procedure:

```
Function f is
    procedure write automaton is
    for each i from 0 to s loop
        interpret i as a configuration of M;
        write couples (i,j) such that i ⊢_M j;
```

```
        end loop
    end write automaton;
begin of f
read w = x_1 x_2 ... x_n;
s := number of possible configurations of M on w;
for each k from 1 to |w| loop
    call write automaton;
    write '$';
    write x_k;
    write '$';
end loop;
call write automaton;
end.
```

Hence RAP is complete.

□

In [KL,80] non-uniform complexity measures are defined by means of "advice functions". The classes of the form C/poly, of the problems decidable by machines of the form C with the aid of a polynomially long advice function, have been characterized by Schöning [Schö,83] as the union of C(S) over all S ∈ SP, under very weak sufficient conditions. The proof does not work directly for our on-line model of computation. However, a similar characterization may be proposed by repeating several times the advice in between each two symbols of the input, in the same way as done above for the automata. If codings of the advice h(|x|) for x is allowed in the following way:

$$h(|x|)x_1h(|x|)x_2h(|x|)\ldots x_nh(|x|)$$

then a similar characterization holds and it can be proven that $Pol_a = NLOG_{on}/poly$.

Diana Schmidt has shown how to apply diagonalizations to complete sets for $NLOG_{on}$ and other similar classes, showing that there exist infinite families of incomparable (with respect to log-space reductions) non-complete sets in $NLOG_{on}$. See [Schm,84].

III. Context-free complexity.

In the previous section we have characterized the languages with polynomial approximations in terms of finite automata. We shall now deal with languages having polynomial approximations in terms of context-free grammars. The size $||G||$ of a context-free grammar G is defined as the number of rules it contains. This measure has been used before in [BCMW,81].

Definition 2. Given a language L, the context-free cost of L is given by

$$cf_L(n) = \min \{ ||G|| / L(G) = L \cap \Sigma^n \}.$$

It is easy to prove that every context-free language L has $cf_L(n) = O(n^2)$. It suffices to construct the intersection of a grammar for L and the n+1 state automaton recognizing Σ^n. Using this complexity measure we define the following complexity class:

$$Pol_{cf} = \{ L / \exists k \in N \text{ with } cf_L(n) = O(n^k) \}.$$

The straight line programs are another way of measuring the complexity of finite functions [BM,75]. These programs have been used by Goodrich, Ladner, and Fischer [GLF,77] to compute finite languages. They introduced the union-concatenation cost, which consists in counting the number of operations needed by a straight-line program, using only unions and concatenations.

More formally, given an alphabet Σ, a straight-line program with unions and concatenations (uc-slp) is defined as a sequence of steps such that:

Step one has the form 1 <--- x, $x \in \Sigma$.

Step i has one of the following two forms:

(a) i <--- x, $x \in \Sigma$;

(b) i <--- j θ k

where j and k are previous steps of the program, and $\theta \in \{ \cup , \cdot \}$.

Given a uc-slp β, we associate a language L_i to each step i of in the following manner:

if i <--- x then $L_i = \{ x \}$;

if i <--- j θ k then $L_i = L_j \; \theta \; L_k$.

For a uc-slp β with k steps, the language L_β generated by β is L_k. Now we can define formally the union-concatenation cost of a language:

Definition 3. For a language L, its union-concatenation cost is given by the function

$$uc_L(n) = \min \{ k \; / \; \text{there is a uc-slp } \beta \text{ with k steps such that } L_\beta = L \cap \Sigma^n \}.$$

With respect to this measure we define the following complexity class:

$$Pol_{uc} = \{ L \; / \; \exists \; k \text{ with } uc_L(n) = O(n^k) \}.$$

It was pointed out in [GLF,77] that uc-slp are closely related to context-free grammars; as a matter of fact it is straightforward to prove that these measures are polynomially related and therefore

$$Pol_{uc} = Pol_{cf}.$$

On the other hand it is easy to show that the set of the palindroms over a two letter alphabet has linear uc cost; hence

$$Pol_{a} \subseteq Pol_{uc}.$$

One more way of characterizing the class Pol_{cf} is in terms of pushdown automata (pda). If $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, then its size $||P||$ is the total number of symbols which are necessary to describe it, i. e.:

$$||P|| = \sum_{t \in \delta} |t| \text{ with } t = (q,u,z) \longmapsto (q',z_1z_2\ldots z_k) \in \delta$$

where $|t| = |quzq'z_1\ldots z_k|$.

It is well known that a language is context-free if and only if it is recognized by a pda. As the size of grammars and the size of equivalent pda can be polynomially related we can define Pol_{cf} in terms of pda, just in the same way as the initial index of section II:

$$Pol_{cf} = \{ L \;/\; \exists\, k \text{ and a family of pda's } P_n,\ n \geq 0, \text{ such that } ||P_n|| \leq n^k \text{ and } L(P_n) = L \cap \Sigma^n \}.$$

The model of uniform computation we shall use in this section is an on-line version of the Auxiliary Pushdown Automata (apda) due to Cook [Co,71] (see also [HU,79]). We consider nondeterministic on-line apda working within log-space and using sparse oracles S. The convention regarding space bounds (markers at both ends of work tapes, no bound on oracle tape) are the same as in the log-space machines of the section II. Of course, no bound is imposed on the pushdown store. Fixed an oracle S, we define the following uniform complexity class:

$$ANLOG_{on}(S) = \{ L \ / \text{ there exists an on-line log-space apda M such that } L = L(M,S) \}.$$

Using this model, we can prove a second equivalence between uniform and non-uniform classes:

Theorem 2. $Pol_{cf} = \bigcup_{S \in SP} ANLOG_{on}(S)$.

Proof. The initial segments of any language accepted by an apda under a sparse oracle can be accepted by a family of pda's of polynomial size. This can be proved exactly as the analogous part of Theorem 1, by including the work tape configurations of the apda, as well as an automaton for the oracle S, in the finite control of the pda's. The pushdown of the apda becomes the pushdown of the pda's.

For the converse, let $L \in Pol_{cf}$. There exists a polynomial $p(.)$ and a family of pda's $P_n = (Q_n, \Sigma, \Gamma_n, \delta_n, q_0, z_0, F_n)$ such that $||P_n|| \leq p(n)$ and $L(P_n) = L \cap \Sigma^n$. Observe that the cardinality of the pushdown alphabet is bounded by $p(n)$. Let s be great enough so that in base s the value $p(n)$ may be written within log n cells. Thus we can encode each symbol in Γ_n as a number of

length log n in base s.

Encode the automata in the oracle S as follows:

$$S = \{\ 0^n\$x\$i\$j\$z\$z_1\$\ldots\$z_k\$u\ /\ (q_j, z_1 \ldots z_k) \in \delta_n(q_i, x, z) \text{ and } u = 1 \text{ iff } j \in F_n\ \}.$$

We construct an apda M with oracle S which accepts L. M will have five work tapes:

1. Tape 1 will contain the current state of the P_n being simulated.

2. Tape 2 will contain the next state of P_n.

3. Tape 3 will contain the top symbol of the pushdown of P_n.

4. Tape 4 will contain the length of the currently applied transition of P_n.

5. Tape 5 will succesively contain the symbols of the right hand side of the currently applied transition of P_n.

The apda performs the following procedure:

```
Begin
read first symbol of input x;
write 0 on tape 1;
write z_0 on tape 3;
push the symbols of z_0 from tape 3 into the pushdown;
while input last loop
    guess j on tape 2;              -- next state
```

```
    write on the query tape 0^n$x$i$j$ ;
                                    -- x is the currently scanned
                                    --   input symbol,
                                    -- i is the contents of tape 1
    pop log n symbols from pushdown to tape 3;
                                    -- top of pushdown
    write the contents of tape 3 on the query tape;
    guess k on tape 4;              -- in base s
                                    -- it is the length of rule
    for i := 1 to k do
        guess z_i on tape 5;        -- in base s
        write z_i$ on the query tape;
        push the symbols of z_i;
        end for;
    guess u = 1 or u = 0;
    write u into the query tape;
                                    -- now the contents of the
                                    --   query tape is:
                                    -- 0^n$x$i$j$z$z_1$...$z_k$u
    query;
    if YES then
        write tape 2 on tape 1;
        read next input symbol x;
    else reject;                    -- wrong guess
    end while;
if last accepted u is 1 then accept else reject
end.
```

This proves the result.

□

Taking deterministic pda's (dpda) for defining the non-uniform complexity measure we can define a similar polynomial class. Taking the deterministic version of apda's, it is possible to characterize in a similar way the non-uniform class as the union over sparse oracles S of $ADLOG_{on}(S)$, the class of the sets decidible by on-line dapda's with oracle S. The proof is similar, but more information has to be encoded on the oracle in order to avoid the nondeterministic guesses in the procedure above. The idea is to put in S prefixes of the codings of the transitions of the dpda's to be simulated, so that these transitions can be constructed deterministically one symbol at each time.

In [GLF,77] it is proved that the set

$$\{ ww \mid w \in \{ 0, 1 \}^* \}$$

has exponential context-free cost. However it is easy to construct a log-space on-line nondeterministic Turing machine (and hence a log-space on-line apda) which accepts its complement. Therefore, we can state the following corollaries:

Corollary 2. $ANLOG_{on}$ is not closed under complementation.

Corollary 3. $ADLOG_{on} \subsetneq ANLOG_{on} \subsetneq P$.

Corollary 4. Neither $NLOG_{on} \subset ADLOG_{on}$ nor $ADLOG_{on} \subset NLOG_{on}$.

Corollary 2 is inmediate when considering the set of squares over a two letter alphabet, as indicated above. Corollary 3 follows from Corollary two, because both $ADLOG_{on}$ and P are closed under complementation. Corollary 4 follows from consideration, first, of the complement of the set of squares, which is in $NLOG_{on}$ but not in $ADLOG_{on}$, and secondly of the set of palindroms with a central separator, which is deterministic context-free and hence in $ADLOG_{on}$, but not in $NLOG_{on}$.

Corollary 3 contrasts the equality among the corresponding off-line classes and P, which was proved by Cook [Có,71].

The classes $ADLOG_{on}$ and $ANLOG_{on}$ possess complete sets. In fact, a construction very similar to the one in the section II allows to define a "pda variant" of the problem RAP which turns out to be complete with respect to on-line log-space reductions. We omit this easy construction.

IV. Boolean circuits complexity.

In this section we shall compare again two non-uniform measures with a uniform measure, and relate the classes by them defined with the ones defined in the previous sections. We shall restrict our attention to languages recognized by deterministic off-line Turing machines with sparse oracles S within polynomial time, P(S); this is the uniform measure for this section.

The first model of non-uniform measure will be the size of straight line programs with union, concatenation, and intersection (uci-slp). This measure is just an extension of the uc-slp where the set of operators is taken as $\theta \in \{\cup, \cdot, \cap\}$ [GLF,77].

Definition 4. For any given language L, its union-concatenation--intersection cost is given by the function

$$uci_L(n) = \min \{ k \;/\; \text{there is a uci-slp } \beta \text{ with k steps such that } L_\beta = L \cap \Sigma^n \}.$$

The second model of non-uniform measures is the circuit-size complexity (also known as combinational complexity). This measure has been known for a long time [Lu,58], [Sa,76].

Let us recall that a combinational circuit over variables $x_1 \ldots x_n$ is like a straight line program where each step i has the form:

(a) $i \leftarrow\!\!- x_j$

(b) $i \leftarrow\!\!- \neg j$

(c) $i \leftarrow\!\!- j \ \Delta \ k$ where $\Delta \in \{ \vee, \wedge \}$.

If $i \leftarrow\!\!- x_j$ then the function calculated at step i is

$$f_i(u_1 \ldots u_n) = u_j.$$

If $i \leftarrow\!\!- j \ \Delta \ k$ then the function calculated at step i is

$$f_i(u_1 \ldots u_n) = f_j(u_1 \ldots u_n) \ \Delta \ f_k(u_1 \ldots u_n).$$

If the combinational circuit β has k steps then the function computed by β is $f_\beta = f_k$.

Definition 5. For a finite language L over the alphabet { 0, 1 }, its boolean complexity is given by the function

$$c_L(n) = \min \{ k \ / \text{ there is a circuit } \gamma \text{ with k steps such that } \forall w \ |w| = n, f_\gamma(w) = 1 \text{ iff } w \in L \}.$$

We can define the following two non-uniform complexity classes:

$$Pol_{uci} = \{ L \ / \ \exists \ k \text{ with } uci_L(n) = O(n^k) \}.$$

$$Pol_c = \{ L \mid \exists k \text{ with } c_L(n) = O(n^k) \}.$$

Using the language of the squares defined in tha last section, Goodrich et al. have shown that $Pol_{uc} \not\subseteq Pol_{uci}$ [GLF,77]. On the other hand Meyer has proved that

$$Pol_c = \bigcup_{S \in SP} P(S).$$

We shall close the link among the above classes by stablishing the equivalence between Pol_c and Pol_{uci}. In one direction it has been already shown in [GLF,77] that there exists a constant k such that $uci_L(n) \leq k(c_L(n)+n)$. We shall prove the converse. First let us present some definitions and a technical lemma.

We say that a finite language L <u>has length n</u> if and only if all the words w in L are of length $|w| = n$, and that L <u>has a length</u> if it has length n for some n. A uci-slp β has <u>coherent lengths</u> if and only if every variable i of β generates a lenguage having a length. We show in the following lemma that we may transform a uci-slp in another having coherent lengths, with small overhead.

<u>Lemma 1.</u> For every uci-slp β which is optimal for computing a language L having length n, there exists a uci-slp γ having coherent lengths, computing L, whose size is $O(n^3)$ times the size of

<u>Proof</u>. We will have in γ variables of the form $\langle i,p \rangle$, for each variable i in β and each $p \leq n$; variable $\langle i,p \rangle$ will compute the words of length p of the language computed by variable i of β.

We construct γ from β in the following way:

(a) If i ⟨--- x is an instruction of β, ⟨i,1⟩ ⟨--- x is an instruction of γ.

(b) If i ⟨--- j $\cup$ k is an instruction of , then for each p add to the following instruction:

1. ⟨i,p⟩ ⟨--- ⟨j,p⟩ $\cup$ ⟨k,p⟩ if $L_j \cap \Sigma^p$ and $L_k \cap \Sigma^p$ are nonempty.

2. ⟨i,p⟩ ⟨--- ⟨j,p⟩ if $L_k \cap \Sigma^p$ is empty.

3. ⟨i,p⟩ ⟨--- ⟨k,p⟩ if $L_j \cap \Sigma^p$ is empty.

Rules of type 2 and 3 can be later eliminated by a renaming of variables.

(c) Intersection is handled in an analoguos way.

(d) If i ⟨--- j.k is an instruction of β, then consider for each p the following set:

$$I_{i,p} := \{ \langle q,t \rangle \;/\; q+t = p,\ L_j \cap \Sigma^q \text{ and } L_k \cap \Sigma^t \text{ are nonempty } \}.$$

For every p with $I_{i,p}$ nonempty add to γ the instruction:

$$\langle i,p \rangle \langle\text{---} \sum_{I_{i,p}} \langle j,q \rangle \langle k,t \rangle$$

previuosly decomposed into less than 2p elementary instructions.

Observe that the optimality of β implies that no intermediate language has length greater than n. The number of instructions increases in this construction within a constant factor of n^3. This yields the desired upper bound.

□

Now we prove that from a uci-slp it is possible to build boolean circuits with small overhead.

Theorem 3. For every language L over the alphabet { 0, 1 },

$$c_L(n) = O(n^5 . uci_L(n)).$$

Proof. Let β be a uci-slp with coherent lengths for $L \cap \Sigma^n$. We construct a circuit over n input gates $x_1 \ldots x_n$ accepting this language.

Gates are numbered $\langle i,p,q \rangle$, $p \leq q \leq n$, i a variable of β. We will manage to obtain output 1 in gate $\langle i,p,q \rangle$ if and only if the word formed by concatenation of the values (0 or 1) of the input gates $x_p \ldots x_q$ is in the language computed by the variable i of β. The output of the circuit will be the output of gate $\langle k,1,n \rangle$, where k is the last variable of β.

Construct the circuit c as follows:

(a) If i <--- 0 is an instruction of β, then add to c the instruction $\langle i,p,p \rangle$ <--- $\neg x_p$ for each $p \leq n$.

(b) If i <--- 1 is an instruction of β, then add to c the instruction $\langle i,p,p \rangle$ <--- x_p for each $p \leq n$.

(c) If $i \leftarrow j \cup k$ is an instruction of β, then add to c the instruction $\langle i,p,q\rangle \leftarrow \langle j,p,q\rangle \vee \langle k,p,q\rangle$ for each p, q such that the language L_i has length q-p+1.

(d) If $i \leftarrow j \cap k$ is an instruction of β, then add to c the instruction $\langle i,p,q\rangle \leftarrow \langle j,p,q\rangle \wedge \langle k,p,q\rangle$ for each p, q such that the language L_i has length q-p+1.

(e) Finally, if $i \leftarrow j.k$ is an instruction of β, then for each p, q such that L_i has length q-p+1 and for each t, $p \leq t < q$, add to c the instruction $\langle i,p,q\rangle \leftarrow \bigvee_t \langle j,p,t\rangle \wedge \langle k,t+1,q\rangle$, previously decomposed into less than q-p elementary instructions.

It is easy to check that the language accepted by the boolean circuit c is the same as the language computed by the uci-slp β. Each variable in β yields at most n^2 variables in c. The result follows.

□

As a corollary of the theorem we stablish the desired result:

Corollary 5. $Pol_{uci} = Pol_c$.

V. Concluding remarks.

In the previous sections we have shown how to characterize several known non-uniform complexity measures in terms of uniform ones; sparse oracles have been allowed to the machines specifying the uniform classes. In this way, the families of sets with polynomial non-uniform measure have been shown to coincide with the ones defined by standard uniform complexity classes relativized to sparse oracles.

Getting together the previous results we obtain the following diagram of the complexity classes we dealt with:

$$
\begin{array}{cccc}
\mathrm{Pol}_c & \mathrm{Pol}_{cf} & & \\
\| & \| & & \\
\mathrm{Pol}_{uci} & \mathrm{Pol}_{uc} & \mathrm{Pol}_a & \mathrm{Pol}_{da} \\
\| & \| & \| & \| \\
\bigcup_{S} \mathrm{P}(S) & \bigcup_{S} \mathrm{ANLOG}(S) & \bigcup_{S} \mathrm{NLOG}(S) & \bigcup_{S} \mathrm{DLOG}(S)
\end{array}
$$

where all the unions are taken over all sparse sets S.

Several variants of the presented results can be easily obtained. For example, it is straightforward to prove that if initial index is measured with bidirectional finite automata, then the polynomial class is the same as LOG_{off} with sparse oracles. This class fulfills the conditions proven sufficient in [Schö,83] for being the same as the corresponding "advice" class $\mathrm{LOG}_{off}/\mathrm{poly}$ in the notation of [KL,80].

Also, a stack may be substituted for the pushdown in the initial index with pda's, and the proof works in this case if the uniform class is defined by auxiliary stack machines. A stack is a pushdown with the additional feature that symbols not in the top can be read, although can not be changed. For a study of the stack automata, see [HU,69a]. For machines with an auxiliary stack, see [Co,71].

Many lines remain open along this line of research. We would like to point out one of them, which will be one of our subjects of research. It is not known whether restricting boolean circuits to gates of fan-out one restricts or not the polynomial non-uniform class. We observe that circuits with fan-out one are somehow similar to propositional formulae: in order to get twice the same result you have to copy the whole synthesizing circuit. Evaluation of fully parenthesized propositional formulae can be done within log-space [Lyn,77]. We conjecture that polynomial size boolean circuits with fan-out one are equivalent to polynomial size propositional formulae. On the other hand, off-line log-space can be shown easily to correspond, modulo sparse oracles, to polynomial size branching programs (see, e.g., [BDFP,83]). Is it true that polynomial size propositional formulae describe exactly the languages which can be recognized by off-line deterministic Turing machines within log-space and with access to a sparse oracle? If not, we propose a second uniform class which possibly corresponds to polynomial formulae: alternating logarithmic time. (This class was suggested by M. Sipser.) Observe that a classical result of Spira (see [Sa,76]) allows to transform a polynomial formula into a circuit of logarithmic depth, hence in a new formula of logarithmic depth. An alternating machine can evaluate such a formula in logarithmic time, provided some kind of "random access" to it. A point remains unclear, however: how to encode the formulae in a sparse oracle? Which kind of "oracle device" is appropriate for a logarithmic time machine? We consider that all those questions are worth to study.

1. [BM,75]
A. Borodin, I. Munro: The computational complexity of algebraic and numeric problems. American Elsivier (1975).

2. [BCMW,81]
W. Bucher, K. Culik, H. Maurer, D. Wotschke: Concise description of finite languages. TCS 14 (1981), 227-246.

3. [BDFP,83]
A. Borodin, D. Dolev, F. Fich, W. Paul: Bounds for width two branching programs. 15 STOC, 87-93.

4. [Co,71]
S. Cook: Characterizations of pushdown machines in terms of time-bounded computers. J. ACM 18 (1971), 4-18.

5. [Ga,83]
J. Gabarro: Funciones de complejidad y su relacion con las familias abstractas de lenguajes. Ph. D. Dissertation, Barcelona 1983.

6. See also:
J. Gabarro: Initial index: a new complexity function for languages. ICALP 83, 226-236.

7. [GHSU,77]
M. Geller, H. Hunt, G. Szymanski, J. Ullman: Economy of description by parsers DPDA's and PDA's. TCS 4 (1977), 143-153.

8. [GLF,77]
G. Goodrich, R. Ladner, M. Fischer: Straight line programs to compute finite languages. Conf. on Theor. Comp. Sci., Waterloo (1977).

9. [Gr,76]
S. Greibach: Remarks on the complexity of nondeterministic counter languages. TCS 1 (1976), 269-288.

10. [HLS,65]
J. Hartmanis, P. Lewis, R. Stearns: Classification of computations by time and memory requirements. Proc. IFIP Cont. 65, 1, 31-35 (1965).

11. [HU,69]
J. Hopcroft, J. Ullman: Some results on tape-bounded Turing machines", J. ACM 3 (1969), 168-177.

12. [HU,69a]
J. Hopcroft, J. Ullman: Formal languages and their relation to automata. Addison-Wesley (1969).

13. [HU,79]
J. Hopcroft, J. Ullman: Introduction to automata theory, languages, and computation. Addison-Wesley (1979).

14. [KL,80]
R. Karp, R. Lipton: Some connections between nonuniform and uniform complexity classes. STOC 80, 302-309.

15. [Lu,58]
O. Lupanov: A method of circuit synthesis, Izv. V. U. Z. Radiof., 1 (1958), 120-140.

16. [Sa,76]
J. Savage: The complexity of computing, Wiley Interscience (1976).

17. [Schm,84]
D. Schmidt: On the complement of one complexity class in another. Submitted for publication.

18. [Schö,83]
U. Schöning: A note on small generators, submitted for publication.