Computing Functions with Parallel Queries to NP

Birgit Jenner
Jacobo Torán

Report LSI–93–8–R

Computing Functions with Parallel Queries to NP *

(Extended Abstract)

Birgit Jenner †
Fakultät für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany

Jacobo Torán ‡
Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona, Spain

Abstract

The class Θ_2^p of languages polynomial time truth-table reducible to sets in NP has a wide range of different characterizations. We survey some of them studing the classes obtained when the characterizations are used to define functions instead of languages. We show that in this way the three function classes $FP_{\|}^{NP}$, FP_{log}^{NP} and FL_{log}^{NP} are obtained. We give an overview about the known relationships between these classes, including some original results. (This report appears in present form in the proceedings of the 8th Annual Conference on Structure in Complexity Theory, San Diego, 1993.)

1 Introduction

The study of nondeterministic computations is a central topic in structural complexity theory. The acceptance mechanism of nondeterministic Turing machines captures important computational problems, and therefore such machines are a good tool to define language classes. However, to define general, i.e., other than 0-1 functions, nondeterministic machines as such are not adequate, and it is not clear how nondeterminism can be exploited to compute functions. Mainly because of this reason, when studying the complexity of a computational problem it is common to consider a decisional version of it, transforming the problem into a set or language, and then studying the complexity of the set instead. Information about the complexity of the set then is used to derive information about the complexity of the function. This is not entirely satisfying since many computational problems are "functional" in nature, and they are not as interesting when considered as decisional problems, for example, in general it seems more useful to find a Hamiltonian tour in a graph than to decide whether the graph has one such tour.

There have been however some ideas on how to use nondeterminism as a resource in order to obtain a model to compute functional problems. If we restrict ourselfs to the polynomial-time context, the following three approaches can be destinguished:

- The nondeterministic machine computing the function is restricted to output only one value for a given input.

For polynomial time this generates the class of (partial) functions NPSV [42]. However, this class does not seem to use the full power of nondeterminism. NPSV does not contain the characteristic function of NP complete problems, unless NP=coNP, and there are only a few examples of functions in NPSV that are not known to be deterministically computable.

*Partially supported by DAAD and Spanish Government (Acción Integrada 131–B, 313–AI–e–es/zk).

†On leave until March 1995, visiting Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya supported by a Habilitationsstipendium of Deutsche Forschungsgemeinschaft (DFG-Je 154 2/1). E-mail: jenner@lsi.upc.es

‡Research partially supported by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II). E-mail: jacobo@lsi.upc.es

- One can define an operator on the set of possible output values (or accepting paths) of a nondeterministic machine.

Important complexity classes have been defined considering such operators, for example #P [44], optP [28], and spanP [25] are defined in this way using the operators number of accepting paths, maximum output value and number of different output values, respectively.

- A deterministic transducer with access to an oracle in NP can be considered.

Here, the function is computed by a particular reduction to a set in NP. This is for example the case of the function class $\mathrm{FP}^{\mathrm{NP}}$ that is in some sense equivalent to optP [28] and contains search versions of all the natural problems in NP.

The function classes defined following the last approach depend heavily on the type of oracle access that the deterministic transducer has. In $\mathrm{FP}^{\mathrm{NP}}$ the deterministic machine can query the oracle in an adaptive way, and this class can be therefore considered as the function analogon to the class Δ_2^{p} of sets Turing reducible to NP.

In this article, we will be interested in a further classification of functions in $\mathrm{FP}^{\mathrm{NP}}$ by considering different kinds of restricted access to the NP oracle. In particular, we investigate functions that are computable when the query mechanism is nonadaptive, that is, when all the oracle queries are made in parallel so that they do not depend on previous oracle answers. In the language case, this restriction gives rise to the class Θ_2^p [45], the non-adaptive analogon to the adaptive class Δ_2^p. Θ_2^p is a very robust class that can be characterized in a wide variety of ways [2], [9], [20], [45]. In Section 2 we survey the characterizations obtained for the language class Θ_2^{p} and show that in the case of functions they give rise to at most three function classes $\mathrm{FP}_{\|}^{\mathrm{NP}}$, $\mathrm{FP}_{\log}^{\mathrm{NP}}$, and $\mathrm{FL}_{\log}^{\mathrm{NP}}$. We obtain a new characterization of $\mathrm{FP}_{\|}^{\mathrm{NP}}$ by single-valued transducers with restricted access to an oracle in NP, that is used in some of the proofs of Section 4. Examples of functions in the three classes are given in Section 3. We exhibit a complete problem for $\mathrm{FP}_{\|}^{\mathrm{NP}}$ that has a natural restriction that makes it complete for $\mathrm{FP}_{\log}^{\mathrm{NP}}$.

In Section 4 we present evidence for the fact that the three function classes are all different. We show that any of the equalities has unlikely consequences. We survey the known consequences of these hypothesis and prove that $\mathrm{FL}_{\log}^{\mathrm{NP}}$ coincides with any of the other two classes if and only if $\mathrm{L} = \mathrm{P}$, and also that if $\mathrm{FP}_{\|}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}}$ then we can obtain a polylogarithmic speed-up in the number of nondeterministic bits needed to compute a problem in NP.

In Section 5 we consider classes of functions computed with a polylogarithmic number of queries to an NP oracle, generalizing the characterizations given in Section 2.

We assume familiarity with basic concepts and notation of structural complexity theory as, e.g., presented in [6]. Any other notion is defined just before it is referred to the first time, or relevant literature is cited. Because of space reasons many of the proofs have been sketcher or ommited. For complete proofs we refer the reader to the final version of the paper.

2 Function classes related to Θ_2^{p}

As mentioned in the introduction, the closure of NP under polynomial-time Turing reducibility defines the language class P^{NP} or Δ_2^{p} (the second level of the Polynomial Hierarchy). Much attention has been devoted to the study of various kinds of restrictions of polynomial-time Turing reducibility, like, e.g.,

- polynomial-time truth-table reducibility or, equivalently, non-adaptive (parallel) reducibility (denoted $\mathrm{P}_{\|}^{\mathrm{NP}}$) [31].

Here a list with all queries is constructed and queried to the oracle, i.e., the queries are made in parallel. The oracle provides the answers to the queries as a 0–1 string denoting the characteristic sequence of the queries.

- polynomial-time reducibility with logarithmically many queries (denoted $\mathrm{P}_{\log}^{\mathrm{NP}}$) [28].

Here the number of queries for any input of length n is bounded by $O(\log n)$.

- Logarithmic-space Turing reducibility (denoted L^{NP}) or, equivalently, logarithmic-space truth-table reducibility or non-adaptive (parallel) reducibility (denoted $\mathrm{L}_{\|}^{\mathrm{NP}}$) [30].

Note that for logspace oracle machines, the query tape is not included in the space bound. Due to the (implicit) time bound of the machine, the length of any query is polynomially bounded. The handling of parallel queries in a logspace computation needs some further explanation. A logspace machine writes a sequence of queries in the oracle tape and receives the oracle answers as a sequence of 0-1 answers (on the same tape). The machine may read these answers in a one-way mode (or, equivalently, two-way mode (see [4])).

- logarithmic-space reducibility with logarithmically many queries (denoted $\mathrm{L}^{\mathrm{NP}}_{\log}$) [45].

- reducibility with logspace-uniform circuits of constant depth (denoted $\mathrm{AC}^0(\mathrm{NP})$) [12], [47].

A language L is contained in $\mathrm{AC}^0(\mathrm{NP})$, if there is a uniform family $\{C_n\}$ of unbounded fan-in circuits with constant depth and size $O(n^{O(1)})$, where C_n is allowed to have oracle nodes for a set $A \in \mathrm{NP}$, such that for all x of length n, $x \in L$ if and only if $C_n(x) = 1$. We assume that there is a deterministic logarithmic-space bounded transducer which on input of 1^n computes an encoding of C_n (logspace uniformity).

Surprisingly, all of these restrictions turned out to be equivalent in the case of NP and give rise to the class Θ^{p}_2 coined by Wagner [45] as the non-adaptive analogon of Δ^{p}_2 in the Polynomial Hierarchy.

Θ^{p}_2 is an extremely robust class that still can be characterized in many other ways (see [2], [9], [20], [45]). In this section we study the mentioned characterizations of Θ^{p}_2 adapting them to compute functions instead of languages. We show that these characterizations generate the three function classes $\mathrm{FP}^{\mathrm{NP}}_{\|}$, $\mathrm{FP}^{\mathrm{NP}}_{\log}$ and $\mathrm{FL}^{\mathrm{NP}}_{\log}$. We have selected from the broad list of ways to define Θ^{p}_2 the above ones since they are particularly interesting in order to ilustrate the different concepts involved in the characterizations, like restricted oracle mechanisms, logarithmic space or circuit complexity. It is not hard to see however that all the characterizations of Θ^{p}_2 given in [2], [9], [20] and [45] than can be adapted in a natural way to define functions, generate one of the three mentioned complexity classes.

The following theorem summarizes those characterizations of Θ^{p}_2 that we use later to define function classes.

Theorem 2.1 [9] [10] [30] [45] *The class* Θ^{p}_2 *of languages truth-table reducible to* NP *can be characterized as* $\Theta^{\mathrm{p}}_2 = \mathrm{P}^{\mathrm{NP}}_{\|} = \mathrm{L}^{\mathrm{NP}}_{\|} = \mathrm{L}^{\mathrm{NP}} = \mathrm{AC}^0(\mathrm{NP}) = \mathrm{P}^{\mathrm{NP}}_{\log} = \mathrm{L}^{\mathrm{NP}}_{\log}$

For the proof of Theorem 2.1, the "census technique" is of particular importance. This technique was developed by Hemachandra [19] for the proof of $\mathrm{P}^{\mathrm{NP}}_{\|} \subseteq \mathrm{P}^{\mathrm{NP}}_{\log}$ [19]. For a set L in $\mathrm{FP}^{\mathrm{NP}}_{\|}$ and an input x, the "census" refers to the number of parallel queries posed by the base machine computing L that are answered positively by the oracle. The idea to decide L with only logartithmically many queries to NP is to compute first the census k for x (by binary search with logarithmically many queries to an oracle in NP), and then make one more query (x, k) to another suitable NP oracle. The last oracle guesses nondeterministically the k queries that are answered positively, checks that the selected queries are the correct ones (by guessing accepting paths for each of these queries in the nondeterministic machine computing them). With the information of the positively answered queries (and therefore also knowing the negatively answered queries) the oracle can then simulate correctly the complete computation of the base machine. The census technique was also used in [22], [39] and [45]. An iterated version of it can also be used to obtain the inclusion $\mathrm{AC}^0(\mathrm{NP}) \subseteq \mathrm{FP}^{\mathrm{NP}}_{\log}$ [10] or the equivalence of k rounds of parallel queries with one round shown in [9]. We apply the technique in several theorems in this section.

We show now that the characterizations of the class Θ^{p}_2 stated in Theorem 2.1 give rise to three function classes.

We obtain functions by considering deterministic Turing transducers instead of (accepting) machines, or, circuit families with an ordered sequence of output nodes instead of just one output node. We denote the corresponding function classes with the prefix "F".

For general functions, the complexity of computing all bits of a function may exceed the complexity of computing any particular bit. If the characterization allows to iterate bit computations, as in the case of the language classes $\mathrm{P}^{\mathrm{NP}}_{\|}$, $\mathrm{L}^{\mathrm{NP}}_{\|}$, L^{NP}, and $\mathrm{AC}^0(\mathrm{NP})$, the resulting function classes coincide.

Theorem 2.2 $\mathrm{FL}^{\mathrm{NP}}_{\log} \subseteq \mathrm{FP}^{\mathrm{NP}}_{\log} \subseteq \mathrm{FP}^{\mathrm{NP}}_{\|} = \mathrm{FL}^{\mathrm{NP}}_{\|} = \mathrm{FL}^{\mathrm{NP}} = \mathrm{FAC}^0(\mathrm{NP})$.

Proof. (from left to right) The inclusion $\mathrm{FL}^{\mathrm{NP}}_{\log} \subseteq \mathrm{FP}^{\mathrm{NP}}_{\log}$ is trivial, because logarithmic space is a restriction of polynomial time.

The second inclusion is proved in the same way as in the language case: all possible query answer sequences of logarithmic length are checked for correctness using polynomially many non-adaptive queries.

We show now the equality of the remaining four classes. Define the different bits of a function f by the set

$$BIT_f := \{\langle x, i, b\rangle \mid b \in \{0,1\},\ f_i(x) = b\},$$

where $f_i(x)$ denotes the ith bit of $f(x)$.

Claim. For $\mathcal{C} \in \{\mathrm{P}_{\|}^{\mathrm{NP}}, \mathrm{L}_{\|}^{\mathrm{NP}}, \mathrm{L}^{\mathrm{NP}}, \mathrm{AC}^0(\mathrm{NP})\}$ it holds: $f \in \mathrm{F}\mathcal{C}$ if and only if $BIT_f \in \mathcal{C}$.

By Theorem 2.1 and the Claim it follows that $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{FL}_{\|}^{\mathrm{NP}} = \mathrm{FL}^{\mathrm{NP}} = \mathrm{FAC}^0(\mathrm{NP})$.

Proof of Claim. The implication from left to right is trivial. For the other implication, let f be such that for any input x of length n, $|f(x)| \leq p(|x|)$ for a polynomial p. Let T be a device of type $\mathcal{C}$ that computes BIT_f. Let T' be a device that checks for all i, $1 \leq i \leq p(|x|)$, $\langle x, i, 0\rangle$ and $\langle x, i, 1\rangle$ for containment in BIT_f by simulation of T. With this information, T' can obtain the length of $f(x)$ easily. It is $l-1$ for the smallest l, $1 \leq l \leq p(|x|)$, such that $\langle x, l, 0\rangle = 0$ and $\langle x, l, 1\rangle = 0$. The ith bit $f_i(x)$ of $f(x)$ satisfies $f_i(x) = b$ if and only if $\langle x, i, b\rangle \in BIT_f$ for $1 \leq i \leq l$. We have to show that T' remains a device of the same type as device T. This is clear for the case L^{NP}. Here simply one bit $f_i(x)$ after the other is computed by simulating T on $\langle x, i, 0\rangle$ and $\langle x, i, 1\rangle$ until both subroutines result negatively. For the cases $\mathrm{P}_{\|}^{\mathrm{NP}}$ and $\mathrm{L}_{\|}^{\mathrm{NP}}$ the simulation of T proceeds in two phases. First, T' computes for all $\langle x, i, b\rangle$ the sequence of parallel queries that T produces for $\langle x, i, b\rangle$. Then, one subroutine after the other is terminated. To be able to resume the computation on the subroutine $\langle x, i, b\rangle$ for the second phase in the case of logarithmic space, T' resimulates T on $\langle x, i, b\rangle$ from the beginning to obtain the correct configuration. Also, T' keeps a counter as to know where in the 0–1 answer string the relevant oracle information can be found. For the case $\mathrm{FAC}^0(\mathrm{NP})$, we combine $\mathrm{AC}^0(\mathrm{NP})$ circuits for all $\langle x, i, b\rangle$, $1 \leq i \leq p(n)$, $0 \leq b \leq 1$ in parallel to a circuit C_n that computes $f(x)$ for x of length n. Let the output nodes of the circuits for $\langle x, i, 0\rangle$ and $\langle x, i, 1\rangle$ become output nodes $2i-1$ and $2i$ of C_n, respectively. Then $y_1 y_2 \ldots y_{2 \cdot p(n)}$ yields a codification of $f(x)$ in which "10" codifies 0, "01" codifies 1, and "00" codifies a padding symbol. □

The three function classes $\mathrm{FP}_{\|}^{\mathrm{NP}}$, $\mathrm{FP}_{\log}^{\mathrm{NP}}$, and $\mathrm{FL}_{\log}^{\mathrm{NP}}$ seem to be all different, since as we will show in Section 4, any equality between them has unlikely consequences. This (possible) difference in behaviour between language classes and function classes is basically due to a communication problem between the oracle and the base machine that occurs when the bound on the length of the function exceeds the bound on the number of bits handed over by the oracle.

Theorem 2.1 can be considered as a result for 0–1 functions. As shown in the following theorem, it remains true for functions with values that are polynomially bounded, i.e., functions f for which exists a polynomial p such that $f(x) \leq p(|x|)$ or equivalently $|f(x)| \leq O(\log |x|)$. For a function class $\mathcal{F}$, we denote by $\mathcal{F}[\log n]$ the subclass of $\mathcal{F}$ formed by polynomially bounded functions.

Theorem 2.3 $\mathrm{FP}_{\|}^{\mathrm{NP}}[\log n] = \mathrm{FL}_{\|}^{\mathrm{NP}}[\log n] = \mathrm{FL}^{\mathrm{NP}}[\log n] = \mathrm{FAC}^0(\mathrm{NP})[\log n] = \mathrm{FP}_{\log}^{\mathrm{NP}}[\log n] = \mathrm{FL}_{\log}^{\mathrm{NP}}[\log n]$.

Proof. Because of Theorem 2.2, it suffices to show $\mathrm{FP}_{\|}^{\mathrm{NP}}[\log n] \subseteq \mathrm{FL}_{\log}^{\mathrm{NP}}[\log n]$. For this, let f be a function in $\mathrm{FP}_{\|}^{\mathrm{NP}}[\log n]$ computed by the transducer T, and apply the census technique mentioned above. First compute the number of parallel queries that are answered positively by T on input x of length n (the census of x). Therefore, use $O(\log n)$ adaptive queries to an oracle in NP, answering for $\langle x, i\rangle$ whether more than i queries of T are answered positive on input x. Note that the census has size $O(\log n)$ and can be stored by T'. It is not hard to see that there exists an NP machine that for the correct census correctly simulates T by guessing the positive queries of T. Thus, with $O(\log n)$ further queries to this oracle any bit of $f(x)$ can be computed. The total number of queries remains $O(\log n)$. □

The class $\mathrm{FP}_{\|}^{\mathrm{NP}}[\log n]$ can furthermore be characterized by optimization and counting functions. It holds $\mathrm{FP}_{\|}^{\mathrm{NP}}[\log n] = \mathrm{optP}[\log n] = \mathrm{spanP}[\log n]$ [25] [28]. (The logarithmic-space analogon of spanP, the function class spanL [3], is hard for #P, however the corresponding restriction spanL$[\log n]$ is a surprisingly easy complexity class that is included in NC^2 and therefore does not seem to be equivalent to $\mathrm{FP}_{\|}^{\mathrm{NP}}[\log n]$).

We show now that similar to Theorem 2.3, if the oracle is a multi-valued function in NP, again all the classes of Theorem 2.1 coincide. Now the communication problem mentioned above is avoided because the oracle provides sufficient bits.

Let us define first the class NPMV [42] of multi-valued functions in NP. A nondeterministic transducer T is a standard nondeterministic Turing machine with additional output tape. We say that $f(x)$ is a value of T, if there is an accepting computation of T on x for which $f(x)$ is the final content of $T's$ output tape. NPMV is the class of all partial, multivalued functions computed by nondeterministic polynomial time-bounded transducers.

We define informally how the oracle function provides information about the multi-valued function for the base machine. Following [16], on querying its oracle $f \in \mathrm{NPMV}$ with the query x, the machine will receive a value $f(x)$ if one exists, and will receive a special symbol $\perp$, otherwise. We refer the reader to [16] for a more formal definition of the classes function that can be computed with the help of an oracle in NPMV.

Theorem 2.4 $\mathrm{FP}_{\|}^{\mathrm{NPMV}} = \mathrm{FL}_{\|}^{\mathrm{NPMV}} = \mathrm{FL}^{\mathrm{NPMV}} = \mathrm{FAC}^0(\mathrm{NPMV}) = \mathrm{FP}_{\log}^{\mathrm{NPMV}} = \mathrm{FL}_{\log}^{\mathrm{NPMV}}$.

Proof. The proof of $\mathrm{FP}_{\|}^{\mathrm{NPMV}} = \mathrm{FP}_{\log}^{\mathrm{NPMV}}$ was obtained in [16]. For the inclusion of the first three classes in $\mathrm{FL}_{\log}^{\mathrm{NPMV}}$ we can apply the census technique. The census of input x corresponds here to the number of queries that are not answered $\perp$ by T. The census can be computed with logarithmically many queries to NPMV. With the correct census information an NP transducer can simulate the whole $\mathrm{FP}_{\|}^{\mathrm{NPMV}}$ (or $\mathrm{FL}_{\|}^{\mathrm{NPMV}}$, $\mathrm{FL}^{\mathrm{NPMV}}$) computation by guessing any possible combination of non-$\perp$ values that T might have received. A particular selection of these values are handed over as an answer, and can be produced as output. For the inclusion $\mathrm{FAC}^0(\mathrm{NPMV}) \subseteq \mathrm{FL}_{\log}^{\mathrm{NPMV}}$, the census computation is iterated to compute the census for each of the constant many levels of the circuit. The remaining inclusions are straightforward. □

A refinement of NPMV to single-valued functions generates NPSV [42]. NPSV is the class of functions for which there exists a single-valued NP transducer computing them. Note that this transducer may not produce any output, but if it does, then the output must be the same on all paths.

Proposition 2.5 [46] $\mathrm{NPSV} \subseteq \mathrm{FP}_{\|}^{\mathrm{NP}}$.

For total functions in NPSV a complete characterization can be obtained by parallel queries or adaptive queries to an oracle in $\mathrm{NP} \cap \mathrm{coNP}$.

Proposition 2.6 *Let f be a total function. Then, it holds* $f \in \mathrm{NPSV} \iff f \in \mathrm{FP}_{\|}^{\mathrm{NP} \cap \mathrm{coNP}} \iff f \in \mathrm{FP}^{\mathrm{NP} \cap \mathrm{coNP}}$.

Proof. It suffices to show the two implications $f \in \mathrm{NPSV} \Rightarrow f \in \mathrm{FP}_{\|}^{\mathrm{NP} \cap \mathrm{coNP}}$ and $f \in \mathrm{FP}^{\mathrm{NP} \cap \mathrm{coNP}} \Rightarrow f \in \mathrm{NPSV}$.

For the first implication, let f be a total function in NPSV computed by the transducer T. Let BIT_f be as in the proof of Theorem 2.2. Clearly, $BIT_f \in \mathrm{NP} \cap \mathrm{coNP}$, since by simulating T on x, it can be decided whether $f_i(x)$ equals 0 or 1 or whether $|f(x)| < i$.

For the second implication, let $f \in \mathrm{FP}^{\mathrm{NP} \cap \mathrm{coNP}}$ be computed by a transducer T with oracle $A \in \mathrm{NP} \cap \mathrm{coNP}$. Let M_A and $M_{\overline{A}}$ be NP machines that accept the set A and its complement, respectively. To compute $f(x)$, an NPSV transducer T' on input x simulates T and for each oracle query q of T, T' guesses the answer 0 or 1 nondeterministically and accordingly starts a subroutine that consists in the simulation of M_A (guessed value 1) or $M_{\overline{A}}$ (guesses value 0) on q. If a subrouting halts in an accepting (rejecting) state, the computation of T will be continued (aborted) by T'. It is obvious that T' is single-valued and outputs $f(x)$. □

We consider NPSV transducers with an oracle in NP to obtain a new characterization of $\mathrm{FP}_{\|}^{\mathrm{NP}}$ that will be very useful in some of the proofs of Section 4. For this we have to restrict the way in which the transducer can access the oracle set.

We say that an NPSV transducer T has restricted access to an oracle A, if all the oracle queries are performed by T before it makes any nondeterministic move. We denote by $\mathrm{rNPSV}^{\mathrm{NP}}$ the class of functions that are computable by an NPSV transducer with restricted access to an oracle in NP, and for a function f, $\mathrm{rNPSV}_f^{\mathrm{NP}}$ denotes the class of functions defined in this way but making at most $O(f)$ queries to the NP oracle.

Note that any computation of an NPMV transducer T with restricted access to an oracle consists of two phases. In the first phase, T works deterministically but has access to the oracle. In the second phase, T may guess but is not allowed to pose further queries. Hence, the complexity of such a transducer in a sense is the complexity of $\mathrm{FP}^{\mathrm{NP}}$ "plus" the complexity of NPSV. By Proposition 2.5, $\mathrm{NPSV} \subseteq \mathrm{FP}_{\|}^{\mathrm{NP}} \subseteq \mathrm{FP}^{\mathrm{NP}}$. Hence, it holds $\mathrm{P}^{\mathrm{NP}} = \mathrm{rNPSV}^{\mathrm{NP}}$. But we furthermore obtain the following characterization of $\mathrm{FP}_{\|}^{\mathrm{NP}}$.

Theorem 2.7 $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{rNPSV}_{\log n}^{\mathrm{NP}}$.

Proof. From left to right, again use the census technique. Let f be a function in $\mathrm{FP}_{\|}^{\mathrm{NP}}$ computed by the transducer T. For input x, compute the number of parallel queries answered positively by $O(\log n)$ queries. Knowing this number, with an NP computation the corresponding queries can be guessed and verified. For the unique correct sequence of query answers, T is simulated and $f(x)$ is produced as output.

For the other inclusion, let T' be a NPSV transducer with restricted access to an oracle $A \in \mathrm{NP}$ and query bound $O(\log n)$ for input of length n. Divide the computation of T' on input x in two parts, up to the configuration c_x just before the first nondeterministic step, and the rest. Computing c_x from x and T' is a function in $\mathrm{FP}_{\log}^{\mathrm{NP}}$, and by Theorem 2.2 it can be computed in $\mathrm{AC}^0(\mathrm{NP})$. Given c_x, the rest of T's computation is a function in NPSV that by Proposition 2.5 can also be computed in $\mathrm{FAC}^0(\mathrm{NP})$. The composition of two functions in $\mathrm{FAC}^0(\mathrm{NP})$ clearly remains in this class and $\mathrm{FAC}^0(\mathrm{NP})$=$\mathrm{FP}_{\|}^{\mathrm{NP}}$ by Theorem 2.2. □

3 Some examples of functions in $\mathrm{FP}_{\|}^{\mathrm{NP}}$

There are important examples of functions in the class $\mathrm{FP}_{\|}^{\mathrm{NP}}$ and in the subclasses considered in the previous section. In some cases we can show that these examples are complete.

To define completeness in a class of functions and to compare the relative complexity of two functions we use the notion of metric reducibility introduced by Krentel in [28].

A function f is metric reducible to a function g if there is a pair of polynomial time computable functions h_1 and h_2 such that for every string x, $f(x) = h_2(x, g(h_1(x)))$. A more intuitive way to to define this notion is to say that f can be computed in polynomial time by a deterministic machine that queries at most once a functional oracle for g.

We start by giving an example of a metric complete function for $\mathrm{FP}_{\|}^{\mathrm{NP}}$. For a boolean formula F on n variables consider the set $Assign(F)$ formed by the strings representing satisfying assignments for F plus the string 0^n. We define the function sup : {Boolean formulas} $\longrightarrow \{0,1\}^*$. $sup(F)$ is the supremum of $Assign(F)$ under the standard lattice partial order (a string $x \in \{0,1\}^n$ is smaller or equal than a string $y \in \{0,1\}^n$ if for every position i, if x has a "1" in this position then so does y).

Theorem 3.1 [24] *The function sup is metric complete for the class* $\mathrm{FP}_{\|}^{\mathrm{NP}}$.

Proof. Let f be a function in $\mathrm{FP}_{\|}^{\mathrm{NP}}$. We will show that f is metric reducible to *sup*. The result follows since clearly $sup \in \mathrm{FP}_{\|}^{\mathrm{NP}}$. We can suppose that the function f is computed by a polynomial time machine M querying the oracle SAT in a non-adaptive way. For some polynomial q on input x, M produces a set of queries $F_1, \ldots, F_{q(|x|)}$ to SAT, and from the oracle answers it computes the value of $f(x)$. Let us suppose that each of the formulas F_i in the list of oracle queries has $l(i)$ variables $x_1^i, \ldots, x_{l(i)}^i$, and that the variables for every pair of formulas are disjoint. From the input x, the formula

$$F_x = \bigwedge_{i=1}^{q(|x|)} (F_i(x_1^i, \ldots, x_{l(i)}^i) \vee (x_1^i = 0 \wedge \ldots \wedge x_{l(i)}^i = 0))$$

can be obtained in polynomial time. Then from $sup(F_x)$ the supremum of each one of the formulas in the sequence of queries can be found and therefore it can be decided which of these formulas are satisfiable. With this information $f(x)$ can be computed in polynomial time. □

One of the most interesting open questions concerning $\mathrm{FP}_{\|}^{\mathrm{NP}}$ is whether the search functions related to NP decisional problems can be computed within this class. For an NP problem L characterized using a polynomial time relation P and a polynomial p so that for every string x

$$x \in L \Longleftrightarrow \exists y,\ |y| \leq p(|x|)\ (x,y) \in A$$

a search function is a function f that for every intance x, if $x \in L$ then $(x, f(x)) \in A$, (if $x \notin L$ then f can take any value).

For the known NP-complete problems like for example SAT, solutions to the search problem (a function that for a boolean formula produces a satisfying assignment for it, in case one exists) can be computed quering an oracle in NP using the self-reducibility properties of the problem. However, the oracle queries made in this process are adaptive and it is not known whether they can be substituted by queries done in some nonadaptive or parallel way. This question is addressed in [1].

To our knowledge, the only problems in NP that are not known to be in P but have search functions in $FP_{\|}^{NP}$ are the Primality and Graph Automorphism problems. The first one is known to be in FewP [15] and it can be easily seen that all the problems in this class have search functions in $FP_{\|}^{NP}$. Another interesting function related to this problem is *factors*, the function that for a number $n \in N$ produces the list of its prime factors. *factors* is even in the class NPSV; this is not hard to see considering that the Primality problem belongs to NP$\cap$ co-NP [38] (in fact the function *factors* can be considered as a search function for Primality). For the case of Graph Automorphism the method to obtain solutions in $FP_{\|}^{NP}$ uses some group theoretic arguments particular to this problem [26], [27]. It has even been shown that solutions to the search problem for Graph Automorphism can be computed making parallel queries to Graph Automorphism itself [34].

Observe that the two mentioned problems are not believed to be NP-complete, and it remains open whether solutions for the search version of NP-complete problems can be found in $FP_{\|}^{NP}$. This question has a positive answer relative to a random oracle as shown in [46].

Theorem 3.2 [46] *Relative to a random oracle the search problem for* SAT *has a solution in* $FP_{\|}^{NP}$.

This contrasts with the following relativized result, obtained in [36].

Theorem 3.3 [36] *There is an oracle under which the search problem for* SAT *has no solution that can be computed in* $FP_{\|}^{NP}$.

Another example of a function in $FP_{\|}^{NP}$ is the one computing the number of isomorphism in two given graphs [35] [27]. This fact is considered as evidence that the Graph Isomorphism problem is not NP-complete since the counting versions of the known NP-complete problems are not even in the polynomial time hierarchy (unless it collapses).

All the given examples are functions in $FP_{\|}^{NP}$ that are not known to be in its subclass FP_{log}^{NP}. Krentel [28] showed that many optimization problems whose solution is polynomially bounded are metric complete for FP_{log}^{NP}. Examples of these problems are the function that computes the maximum size of a clique in a graph, or the one obtaining the maximum number of simultaneously satisfiable clauses in a boolean formula written in conjunctive normal form. A function of this kind related to the function *sup* presented above, is the function *sup'* that for a boolean formula F, computes the number of "1's" in $sup(F)$.

Theorem 3.4 *The function sup' is metric complete for the class* FP_{log}^{NP}.

All these examples of complete functions in FP_{log}^{NP} belong also to the class FL_{log}^{NP}. It does not make sense to talk about metric completeness for this last class since the closure of FL_{log}^{NP} under polynomial time metric reducibility coincides with FP_{log}^{NP}. However, as we will see in the next section FL_{log}^{NP} seems to be a much weaker class than FP_{log}^{NP}. In particular "hard" functions in FP (like the Circuit Value function [29]) probably do not belong to FL_{log}^{NP}.

The question of whether search functions for NP-complete problems belong to FP_{log}^{NP} (or FL_{log}^{NP}) is equivalent to the P versus NP question. This follows from a result by Krentel [28] that shows (stated in a slightly different way) that if the search version for a problem A in NP has a solution in FP_{log}^{NP} then $A \in P$.

4 Consequences of the equality of the function classes

In this section we present evidence of the fact that the three function classes $\mathrm{FP}_{\|}^{\mathrm{NP}}$, $\mathrm{FP}_{\log}^{\mathrm{NP}}$ and $\mathrm{FL}_{\log}^{\mathrm{NP}}$ are all different. We compare first the two classes that seem to be weaker.

Theorem 4.1 $\mathrm{FP}_{\log}^{\mathrm{NP}} = \mathrm{FL}_{\log}^{\mathrm{NP}}$ *if and only if* $\mathrm{P} = \mathrm{L}$.

Proof. From left to right, let *cceval* denote a complete circuit evaluation function that, for a boolean circuit C and x, computes the value of each gate of $C(x)$. Clearly, *cceval* $\in \mathrm{FP}$, and hence *cceval* $\in \mathrm{FP}_{\log}^{\mathrm{NP}}$. Now, suppose that *cceval* $\in \mathrm{FL}_{\log}^{\mathrm{NP}}$, and let T be a log space transducer that computes *cceval* with $O(\log n)$ many queries to SAT, for an input circuit C and x of size n. We will show that then the circuit value problem (which is P-complete [29]) can be computed in log space as follows. On input of C and x, a log space transducer T' cycles through all possible answer sequences y of length $O(\log n)$ and simulates T following the answer sequence y. For each y, T' checks that the output produced by T is a correct sequence of gates of C and that all values attached to the gates are correct. To achieve this, for any gate g T' repeats the simulation of T to find the values of the (at most two) inputs of g. When an answer sequence y is tested, for which all the values of the circuit are correct, again by resimulation on y, T' looks up the value of the output gate of C and rejects or accepts accordingly.

From right to left, note that $\mathrm{P} = \mathrm{L}$ if and only if $\mathrm{FP} = \mathrm{FL}$. Suppose that $\mathrm{FP} \subseteq \mathrm{FL}$, and let T be a transducer with oracle $A \in \mathrm{NP}$ that computes a function $f \in \mathrm{FP}_{\log}^{\mathrm{NP}}$. Let $c \cdot \log n$ be the bound on the number of queries for an input x of length n. First, note that only logarithmic space and maximally $2 \cdot c \cdot \log n$ queries to the following oracle $A'' \in \mathrm{NP}$ are neccessary to obtain the correct query answer sequence of T on input x.

$$A'' := \{\langle x, p\rangle \mid p \in \{0,1\}^*, \text{ the } p+1\text{st query in the computation of } T \text{ on } x, \text{ with } p \text{ taken as prefix of the query answer sequence, is answered positively}\}.$$

Now, define the function f'' with

$$f''(\langle x, y\rangle := \text{output that } T \text{ produces on input } x, \text{ if } y \in \{0,1\}^{c\cdot\log n} \text{ is taken as the sequence of query answers}$$

Clearly, $f'' \in \mathrm{FP}$, and by assumption $f'' \in \mathrm{FL}$. Hence, f can be computed with logarithmic space and $O(\log n)$ queries to A'', i.e., $f \in \mathrm{FL}_{\log}^{\mathrm{NP}}$. □

From the proof of the above theorem follows also that even the hypothesis $\mathrm{FP} \subseteq \mathrm{FL}_{\log}^{\mathrm{NP}}$ would imply $\mathrm{L} = \mathrm{P}$. Also as a consequence of Theorem 4.1, it seems unlikely that parallel queries to NP can be reduced to logarithmic adaptive queries with logarithmic space.

Corollary 4.2 *If* $\mathrm{FL}_{\|}^{\mathrm{NP}} \subseteq \mathrm{FL}_{\log}^{\mathrm{NP}}$, *then* $\mathrm{L} = \mathrm{P}$.

The equality of the classes $\mathrm{FP}_{\|}^{\mathrm{NP}}$ and $\mathrm{FP}_{\log}^{\mathrm{NP}}$ would also imply strong consequences. We show first that the question of whether the classes are equal can be characterized using Generalized Kolmogorov Complexity and the concept of polynomial enumerators. We define these notions.

Generalized Kolmogorov complexity measures how far a string can be compressed and how fast a string can be recomputed from its compression (cf [32]). A function f has low Kolmogorov Complexity relative to the input ($f \in K[\log, \mathrm{poly} \mid x]$) [18] if there is a constant c such that for every x in the domain of f there is a string y (the compression of $f(x)$) of size at most $c\log(|x|)$ such that the Universal Machine on input (x, y) prints $f(x)$ in at most $|x|^c$ steps.

Polynomial enumerators for functions have been introduced in [11] as a model of function approximation. A function f has a polynomial enumerator if there is a polynomial time machine that for an input x outputs a list of (polynomially many) values, one of which is the correct value $f(x)$.

The next result relates both concepts.

Proposition 4.3 *Let f be a function, the following statements are equivalent:*

1. $f \in K[\log, \text{poly} \mid x]$.
2. *f has a polynomial enumerator.*
3. *For some oracle A, $f \in \text{FP}^A_{\log}$.*

The equivalence on 1 and 2 has been shown in [17]. And the equivalence of 2 and 3 is from [5]. Lozano [33] has observed that for the particular case of the functions in the class $\text{FP}^{\text{NP}}_{\|}$ it sufficies that the set A is statement 3 belongs to NP, and therefore the question $\text{FP}^{\text{NP}}_{\|} = \text{FP}^{\text{NP}}_{\log}$ is equivalent to whether every function in $\text{FP}^{\text{NP}}_{\|}$ has low Kolmogorov Complexity or every function in the class has a polynomial enumerator.

We first show that if the classes $\text{FP}^{\text{NP}}_{\|}$ and $\text{FP}^{\text{NP}}_{\log}$ coincide then there is a polynomial time algorithm that decides corectly the satisfiability of a formula with at most one satisfying assignment. (If the formula has more than one assignment the algorithm can incorectly decide that the formula is not satisfiable). This result is stated formally using the concept of promise problems (see [14]). A promise problem is a pair of sets (Q, R). A set L is called a solution to the promise problem (Q, R) if

$$\forall x(x \in Q \Rightarrow (x \in L \Leftrightarrow x \in R)).$$

1SAT denotes the set of boolean formulas with at most one satisfying assignment.

Theorem 4.4 *If $\text{FP}^{\text{NP}}_{\|} \subseteq \text{FP}^{\text{NP}}_{\log}$, then the promise problem* (1SAT, SAT) *has a solution in* P.

Proof. (Sketch) Consider the function *sup* that computes the supremum of the set of satisfying assignments of a boolean formula (see 3.1). *sup* $\in \text{FP}^{\text{NP}}_{\|}$ and if the hypothesis holds by proposition 4.3 there is a polynomial enumerator that for a formula F produces a list of polynomially many possible values for *sup*(F). We can construct an algorithm that on input F runs the enumerator and accepts if and only if at least one of the produced values is a satisfying assignment for F. It is clear that the algorithm decides correctly unsatisfiable formulas and also those formulas that can be satisfied by their own supremum, which include the formulas with just one satisfiable assignment. □

If the promise problem (1SAT, SAT) has a solution in P then the next unexpected equalities would follow. The complexity classes FewP and R mentioned in the result are well known and we refer to the standard literature for definitions. US [8] is the class of languages computed by a polynomial time nondeterministic Turing machine that accepts an input if it produces exactly one accepting path.

Theorem 4.5 *If the promise problem* (1SAT, SAT) *has a solution in* P *then* FewP=P, NP=R *and* coNP=US.

The first two equalities have been obtained in [43]. The theorems above can be summarized as in the next corollary. The result written in this form is from [40] although some of the ideas behind it are already considered in another context in [7].

Corollary 4.6 *If $\text{FP}^{\text{NP}}_{\|} \subseteq \text{FP}^{\text{NP}}_{\log}$, then* FewP=P, NP=R *and* coNP=US.

A polynomial time algorithm for the promise problem for (1SAT, SAT) would imply also consequences of a different flavour, namely that the Graph Automorphism and the Primality problems could be solved in polynomial time. The result for Graph Automorphism was obtained in [26], (see also [27]); for the second problem the result follows from theorem 4.4 and the fact that the Primality problem is contained in UP and therefore in FewP [15].

Corollary 4.7 *If $\text{FP}^{\text{NP}}_{\|} \subseteq \text{FP}^{\text{NP}}_{\log}$, then the Graph Automorphism and the Primality problems are in* P.

We present now a different consequence of the equality of the function classes that contrary to the previous results, does not seem to be related with the promise problem (1SAT, SAT). We show that if $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ then a polylogarithmic amount of nondeterminism can be simulated in polynomial time and also SAT can be decided (for any k) in polynomial time with the help of only $O(\frac{n}{log^k n})$ nondeterministic bits. This still does not imply P=NP but in some sense makes "smaller" the gap between the classes

Subclasses of NP with bounded nondeterminism have been considered in [23], [13], (see also [37] in these proceedings). For a function f we denote by $\mathrm{NP}(f)$ the subclass of NP formed by the languages $L \in \{0,1\}^*$ for which there is a set $A \in P$ and a constant $c \in \mathbf{N}$ such that for every string x

$$x \in \mathrm{L} \Longleftrightarrow \exists y,\ |y| \leq cf(|x|),\ (x,y) \in A.$$

To prove our result we need the following theorem:

Theorem 4.8 *If* $\mathrm{FP}_{\|}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then there is a function* $f \in r\mathrm{NPSV}_{[\log \log]}^{\mathrm{NP}}$ *that for a sequence of boolean formula* $F_1, \ldots, F_n$ *outputs one satisfiable formulas from the list in case one exists. (If all the formulas are unsatisfiable then the value of* f *is some special symbol).*

Proof. (Sketch) Consider the function g that for a sequence of boolean formulas $F_1, \ldots, F_n$ outputs its characteristic vector, that is the string $v = a_1 a_2 \ldots a_n$ with $a_i = 1$ if $F_i \in \mathrm{SAT}$ and $a_i = 0$ otherwise. Clearly $g \in \mathrm{FP}_{\|}^{\mathrm{NP}}$ and using the hypothesis and proposition 4.3 there is a polynomial enumerator for g. Running the enumerator on $F_1, \ldots, F_n$ one can obtain a list L of polynomially many potential values for g, $L = (v_1, \ldots, v_{p(n)})$ one of which is the correct one. We can asume that at least one of the formulas is satisfiable (this can be checked with just one query to NP), and therefore if the string 0^n appears in this list, it can be deleted.

We explain how to identify the values of the list with the nodes of a directed acyclic graph (DAG) in such a way that a correct "1" in the characteristic vector of $F_1, \ldots, F_n$ can be identified with just $O(\log \log n)$ queries to NP. For this we construct a DAG $G = (V, E)$ where $V = \{v_1, \ldots, v_{p(n)}\}$ and $E = \{(v_i, v_j) \mid v_j \prec v_i\}$ ($v_j \prec v_i$ means that v_j is smaller than v_i in the lattice order, that is, for every position l, if v_j has a "1" in position l then so does v_i).

We contract G to obtain a simpler graph G_1 using the next three rules.

- If v is a node without descendants (a leaf) and v has more than one "1" then turn to "0" all the "1's" in v except the first one.
- Denote every node v by the supremum of the leafs that can be reached from v
- Contract all the nodes that have the same value into a single node.

We define the level of a node as the number of "1's" it has. Also we will say that a node v is consistent with the characteristic vector of $F_1, \ldots, F_n$ (or just that v is consistent) if v is smaller than or equal (in the lattice order) than this value. In a consistent node all the "1's" from its value correspond to satisfiable formulas in the sequence $F_1, \ldots, F_n$. G_1 satisfies the following properties:

i. If v is node of level k in G_1 then v reaches exactly k leafs.

ii. If k is the maximum level in which there is a node of level k consistent with the characteristic vector of $F_1, \ldots, F_n$ then there is exactly one such node at level k (this node is the supremum of the consistent leafs in G_1).

iii. In G_1 there is at least one node consistent with the characteristic vector of $F_1, \ldots, F_n$.

Because of properties *ii* and *iii*, it suffices to obtain the maximum level with a consistent node in G_1. Once the level is known, the unique consistent node in this level can be obtained nondeterministically by guessing a node with k "1's" and checking that for any of the "1's" the corresponding formulas in the sequence $F_1, \ldots, F_n$ have some satisfying assignment.

We intend to obtain the maximum level of a consistent node with binary search querying an oracle in NP. Observe however that the graph G_1 can have as many as n levels and therefore $O(\log n)$ queries to NP seem to be needed. In order to make fewer queries we make use of the following claim.

Claim: Let p be a polynomial and A_p be the set of pairs (G, n) where $n \in \mathbf{N}$ and G is a (contracted) DAG formed from a set of strings in $\{0,1\}^n$ as explained above, with at most $p(n)$ nodes. There is a polynomial time

algorithm that (for sufficiently large n), on input a pair $(n, G) \in A_p$ with m leafs, selects a set S of at most $\frac{m}{2}$ leafs of G in such a way that every node in G with at least $\log^4 n$ leaf descendants can reach some leaf in S.

The algorithm of the claim can be applied to (n, G_1) (observe that since the size of the list of values is polynomially bounded by p, G_1 has at most $p(n)$ nodes), obtaining a set S of at most $\frac{n}{2}$ leafs. We construct a new graph G_2 by turning to "0" the positions of the nodes in G_1 corresponding to the leafs that are not in S, and then applying the contraction rules to this graph. Intuitively if we turn some positions to "0" we are throwing aside the formulas in the input sequence that correspond to these positions and consider only those formulas corresponding to "1" positions.

If in G_1 there is a node at level $k \geq \log^4 n$ consistent with the correct value, then G_2 satisfies all three properties of G_1, (property *iii* is satisfied since by the claim the node at level k has some leaf descendant in S, and this leaf is a consistent node in G_2). Additionally G_2 has at most $\frac{n}{2}$ leafs and therefore at most $\frac{n}{2}$ levels.

The algorithm of the claim is applied successively to G_2 to obtain G_3 and so on, until a graph G_i ($i \leq \log n$) with no node at a level $k \geq \log^4 n$, is obtained.

Given the collection of graphs $G_1, \ldots, G_i$, in one of the graphs a node consistent with the correct value of $g(F_1, \ldots, F_n)$ can be found with only $O(\log \log n)$ queries to NP in the following way: First with $O(\log \log n)$ queries to an NP set obtain the largest j for which there is a graph G_j with a node consistent with the correct value of the characteristic vector of $F_1, \ldots, F_n$ at a level $l \geq \log^4 n$. Then obtain the largest level k in graph G_{j+1} with a node consistent with the correct vector. This can be done again using binary search with only $O(\log \log n)$ queries to an NP set since $k < \log^4 n$. In level k of graph G_{j+1} there is just one consistent node, and therefore once j and k are known, this node can be obtained nondeterministically. Each "1" in the node corresponds to a satisfiable formula in the input sequence. One can select for example the formula corresponding to the first "1" in the node. It follows that the claimed function f is in $\mathrm{rNPSV}^{\mathrm{NP}}_{[\log \log n]}$

Proof of the Claim. (Sketch) Let p be a polynomial and G be a graph with m leafs and $(G, n) \in A_p$. We show first that (for sufficiently large n) there is a set S' of $\frac{m}{\log^2 n}$ leaves with the property that every node from level $k \geq \log^4 n$ can reach at least one leaf in S'. Suppose that this were not true. Then for everycombination C of $m - \frac{m}{\log^2 n}$ leafs there must be a node v at a level $k \geq \log^4 n$ such that all the leaves that can be reached from v belong to C. We say in this case that v is a *bad* node for C. Each node v can be *bad* for at most

$$\binom{m - \log^4 n}{m - \frac{m}{\log^2 n} - \log^4 n}$$

combinations of $m - \frac{m}{\log^2 n}$ leafs. This is because v has at least $\log^4(n)$ leaf descendants and all these leafs must be in the combination. There are $\binom{m}{m - \frac{m}{\log^2 n}}$ combinations of $m - \frac{m}{\log^2 n}$ leafs. If b is the number of nodes in levels higher than $\log^4 n$ we have

$$\binom{m - \log^4 n}{m - \frac{m}{\log^2 n} - \log^4 n} \times b \geq \binom{m}{m - \frac{m}{\log^2 n}}$$

and from this follows $b \geq 2^{\log^2 n}$ which is a contradiction (for sufficiently large n) since $b \leq p(n)$.

The problem to obtain a set of leafs of minimum size that "covers" each node at level $k \geq \log^4 n$ is an instance of the Set Cover problem which is NP-hard. However, it has been shown by Johnson [21] that the greedy algorithm obtains an approximated solution for this problem of size within a logarithmic factor of the optimum. Since we have just shown that there must be a a set of at most $\frac{m}{\log^2 n}$ leafs covering all the nodes at levels higher than or equal to $\log^4 n$, the greedy algorithim obtains in polynomial time a cover of size $\frac{m}{\log n} > \frac{m}{2}$ (for sufficiently large n). □

An interesting observation is that the satisfiable formula in the list $F_1, \ldots, F_n$ selected by the function $f \in \mathrm{rNPSV}^{\mathrm{NP}}_{[\log \log n]}$ in the proof is not necessarily the first one in the sequence, the selection depends on the enumerator for the function g.

Lemma 4.9 *If* $\mathrm{FP}^{\mathrm{NP}}_{\|} \subseteq \mathrm{FP}^{\mathrm{NP}}_{\log}$ *then for any* $k \geq 1$ *there is a function* $f_k \in \mathrm{FP}^{\mathrm{NP}}_{\|}$ *that for a satisfiable boolean formula* F *on* n *variables produces an assignment for the first* $\frac{\log^k n}{(\log \log n)^{k-1}}$ *variables that can be extended to a satisfying assignment of* F.

Proof. (Sketch) We use the characterization $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{rNPSV}_{[\log n]}^{\mathrm{NP}}$. The proof is by induction on k. For $k = 1$ the result is straightforward. For the induction step, let us suppose that an assignment for the first $\frac{\log^k n}{(\log\log n)^{k-1}}$ variables of a satisfiable formula F can be obtained by a function in $\mathrm{FP}_{\|}^{\mathrm{NP}}$, and that this function has a polynomial enumerator. Running the enumerator on F we obtain a list of possible values for such a partial assignment. Substituting these values in F and reducing the formulas, a list of boolean formulas $F_1, \ldots, F_{p(n)}$ is obtained. By the same argument as in Theorem 4.8, a satisfiable one F_i (corresponding to a correct partial assignment of the first variables) can be uniquely selected in a nondeterministic way with the help of $O(\log\log n)$ many queries to an NP oracle. The nondeterministic part of the computation in the algorithm is just needed to output the formula. However, once F_i is determined, the process can be repeated to this formula without having to output it, by making a second round of $O(\log\log n)$ queries to an NP oracle. The queries in the second round have encoded the information obtained in the first round so that the oracle can nondeterministically obtain the formula F_i from this information. With the second round of queries an assignment for the next $\frac{\log^k n}{(\log\log n)^{k-1}}$ variables of the initial formula is obtained. The process can be repeated $\frac{\log n}{\log\log n}$ times until $\log n$ oracle qeries are made. Since in each round $\frac{\log^k n}{(\log\log n)^{k-1}}$ variables are assigned, the total number of assigned variables is $\frac{\log^{k+1} n}{(\log\log n)^k}$. After all the queries are made, a singled valued nondeterministic computation outputs the obtained partial assignment. □

Based on these results we obtain:

Corollary 4.10 *If* $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then for any function* $f : \mathrm{N} \longrightarrow \mathrm{N}$ *and any* $k \in \mathrm{N}$, $\mathrm{NP}(f) \subseteq \mathrm{NP}(\frac{f}{\log^k})$.

Corollary 4.11 *If* $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then for any* $k \in \mathrm{N}$ *the class* $\mathrm{NP}(\log^k)$ *is included in* P.

Corollary 4.12 *If* $\mathrm{FP}_{\|}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then for any* $k \in \mathrm{N}$ $\mathrm{SAT} \in \mathrm{DTIME}(2^{n/log^k n})$, *and* $\mathrm{SAT} \in \mathrm{NP}(\frac{n}{log^k n})$.

It is an open question whether these results can be improved to show that $\mathrm{FP}_{\|}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}}$ implies P=NP.

5 Polylogarithmic Bounds

In this section we state some observations about function classes with polylogarithmic query bounds to an NP oracle. These classes are generalizations of the function classes $\mathrm{FP}_{\log}^{\mathrm{NP}}$ and $\mathrm{FAC}^0(\mathrm{NP})$=$\mathrm{FP}_{\|}^{\mathrm{NP}}$(Theorem 2.2).

Denote $\mathrm{AC}^k(NP)$ the class of languages computable by logspace-uniform families $\{C_n\}$ of unbounded fan-in circuits with constant depth and size $O(n^{O(1)})$ that contain oracle nodes for a set in NP.

Recently, the following generalization of Theorem 2.1 has been obtained by Castro and Seara.

Theorem 5.1 [10] *For any* $k \geq 0$, $\mathrm{AC}^k(\mathrm{NP}) = \mathrm{P}_{\log^{k+1}}^{\mathrm{NP}}$.

Note that there is also a characterization in terms logarithmic space possible, if so-called "adaptive" logspace reductions are used (see [4]).

As one might suspect, the corresponding function classes $\mathrm{FAC}^k(\mathrm{NP})$ and $\mathrm{FP}_{\log^{k+1}}^{\mathrm{NP}}$ do not seem to be equal for any $k > 0$. But we can obtain a characterization of $\mathrm{FAC}^k(\mathrm{NP})$ that shows that a possible "difference" in complexity between $\mathrm{FAC}^k(\mathrm{NP})$ and $\mathrm{FP}_{\log^{k+1}}^{\mathrm{NP}}$ hinges on the "difference" of complexity between $\mathrm{FP}_{\log}^{\mathrm{NP}}$ and $\mathrm{FP}_{\|}^{\mathrm{NP}}$, only.

Proposition 5.2 *For any* $k \geq 0$, $\mathrm{FAC}^k(\mathrm{NP}) = FP_{\log^k}^{\mathrm{FP}_{\|}^{\mathrm{NP}}}$.

Proof. From left to right, let f be a function in $\mathrm{FAC}^k(\mathrm{NP})$ that is computed by the circuit family C_n with oracle nodes for a set $A \in \mathrm{NP}$. C_n has size $p(n)$ for a polynomial p and depth $c \cdot \log^k$ for a constant c. Let $g(\langle x, v_1, \ldots v_{p(n)}\rangle) = \langle v'_1, \ldots, v'_{p(n)}\rangle$ be a function that computes, for input x and value v_i for each gate g_i, new values v'_i (obtained using the input values v for g_i) for all gates of C_n. Clearly, $g \in \mathrm{FP}_{\|}^{\mathrm{NP}}$. With g as oracle C_n

can be evaluated by setting initially $v_1 v_2 \ldots v_n = x_1 x_2 \ldots x_n = x$ and the rest of the gates to 0, and querying $g\ c \cdot \log^k$ times, each time using the new value vector for the next query. For the vector obtained with the i-th query all gates up to depth i carry their correct values. $f(x)$ can be obtained in this way after the last query.

From right to left, by iteration of the census technique. Let f be a function computed by the transducer T_f with $O(\log^k n)$ queries to an oracle $g \in \mathrm{FP}_{\|}^{\mathrm{NP}}$. Let T_g be a transducer that with parallel queries to oracle $A \in \mathrm{NP}$ that computes g. Let $census_A(y)$ be the number of parallel queries answered positively by T_g on input y, and let $y_1, y_2, \ldots, y_{c \cdot \log^k n}$ be the queries of T_f on input x of length n. It is not hard to show that given x and $census_A(y_1), \ldots, census_A(y_i)$ with $O(\log n)$ queries to an oracle in NP $census_A(y_{i+1})$ can be computed, and by Theorem 2.2 this can be done in $\mathrm{AC}^0(\mathrm{NP})$. Thus, to compute $f(x)$, simply combine $c \cdot \log^k n$ $\mathrm{AC}^0(\mathrm{NP})$ circuits (above each other) to obtain the census for all the queries of T on input x, and provide one further layer of oracle nodes to compute each bit of $f(x)$. □

The proof shows that, in fact, we can characterize the classes $\mathrm{AC}^k(\mathrm{NP})$ in terms of NPSV transducers making $O(\log^k n)$ (restricted) queries to an oracle in NP, i.e., it holds $\mathrm{AC}^k(\mathrm{NP}) = \mathrm{rNPSV}^{\mathrm{NP}}_{[\log^k n]}$ for all $k \geq 0$.

A similar characterization as Proposition 5.2 can be shown for the $\mathrm{FP}^{\mathrm{NP}}_{\log^k}$ hierarchy.

Proposition 5.3 *For any* $k \geq 0$, $\mathrm{FP}^{\mathrm{NP}}_{\log^{k+1}} = \mathrm{FP}^{\mathrm{FP}^{\mathrm{NP}}_{\log}}_{\log^k}$.

As a direct consequence of the above decomposition properties of the two hierarchies we obtain the following statement.

Corollary 5.4 *If* $\mathrm{FP}_{\|}^{\mathrm{NP}} \subseteq \mathrm{FP}^{\mathrm{NP}}_{\log}$ *then for every* $k \geq 0$, $\mathrm{FAC}^k(\mathrm{NP}) = \mathrm{FP}^{\mathrm{NP}}_{\log^{k+1}}$.

Acknowledgement

We would like to thank J.L. Balcázar, R. Gavaldá, J. Köbler, T. Lozano, P.B. Miltersen and C. Wilson for interesting discussions on the topics of the paper.

References

[1] K. Abrahamson, M. Fellows and C. Wilson, Parallel self-reducibility, Proc. of the Fourth IEEE International Conference on Computing and Information, May 1992, pp. 67–70.

[2] E. Allender and C. Wilson, Width-bounded reducibility and binary search over complexity classes Proc. 5th IEEE *Structure in Complexity Theory Conference* (1990), pp. 112–130.

[3] C. Àlvarez and B. Jenner, A Very Hard Log-Space Counting Class, *Theoretical Computer Science* **107**,1 (1993), pp. 3–30. Proc. 5th IEEE *Structure in Complexity Theory Conference* (1990), pp. 154–168.)

[4] C. Àlvarez, J.L. Balcázar and B. Jenner, Functional Oracle Queries As a Measure of Parallel Time, Proc. 8th STACS Conference, *Lecture Notes in Computer Science* **480** (Springer, Berlin, 1991), pp. 422–433. (to appear in *Mathematical Systems Theory* as "Adaptive logspace reducibility and parallel time")

[5] A. Amir, R. Beigel, B. Gasarch, Some connections between bounded query classes and non-uniform complexity. Proc. 5th IEEE *Structure in Complexity Theory Conference* (1990), pp. 232–244.)

[6] J.L. Balcázar, J. Diaz and J. Gabarró, Structural Complexity Theory I, Springer-Verlag, Berlin, 1988.

[7] R. Beigel, Bounded queries to SAT and the Boolean Hierarchy, Tech. Report 87-8 (1987) The Johns Hopkins University.

[8] A. Blass and Y. Gurevich, On the unique satisfiability problem, *Information and Control* **55** (1982) pp. 80–88.

[9] S. R. Buss and L. Hay, On thruth-table reducibility to SAT and the difference hierarchy over NP, Proc. 3rd IEEE *Structure in Complexity Conference*, 1988, pp. 224–233.

[10] J. Castro and C. Seara, Characterizations of some complexity classes between Θ_2^{p} and Δ_2^{p}. Proc. 9th STACS, *Lecture Notes in Computer Science* **577** (Springer, Berlin, 1992), pp. 305–319.

[11] J. Cai and L. Hemachandra, Enumerative counting is hard *Information and Computation*, **82** (1989), pp. 34–44.

[12] A.K. CHANDRA, L. STOCKMEYER AND U. VISHKIN, Constant Depth Reducibility, *SIAM Journ. of Comput.* **13**, 2 (1984), pp. 423–439.

[13] J. DÍAZ AND J. TORÁN, Classes of bounded nondeterminism, *Mathematical Systems Theory,* **23** (1990), pp. 21–32.

[14] S. EVEN, A. SELMAN AND Y. YACOBI, The complexity of promise problems with applications to public-key cryptography, *Information and Control,* **61** (1984), 114–133.

[15] M.R. FELLOWS AND N. KOBLITZ Self-witnessing polynomial-time complexity and prime factorization. Proc. 7th IEEE Structure in Complexity Theory Conference (1992), pp. 107–110.

[16] S. FENNER, S. HOMER, M. OGIWARA, AND A. L. SELMAN, On Using Oracles That Compute Values, Proc. 10th STACS, *Lecture Notes in Computer Science* **665** (Springer, Berlin, 1993), pp. 398–408.

[17] F. GREEN AND J. TORÁN, Kolmogorov complexity of #P functions, Internal report LSI-91-2 (1991) Universitat Politécnica de Catalunya.

[18] J. HARTMANIS, Generalized Kolmogorov complexity and the structure of feasible computations, Proc. 24th FOCS conference (1989), pp. 281–286.

[19] L. HEMACHANDRA, The strong exponentiell hierarchy collapses, Proc. 19th ACM Symposium on Theory of Computing (STOC), (1987), pp. 110-122.

[20] B. JENNER, B. KIRSIG AND K.-J. LANGE, The logarithmic alternation hierarchy collapses $A\Sigma_2^L = A\Pi_2^L$, *Information and Computation* ,1 (1989), pp. 269–288.

[21] D.S. JOHNSON, Approximation algorithms for Combinatorial Problems, *Journal of Computer and Systems Sciences* **9** 3, (1974) 256–278.

[22] J. KADIN, $P^{NP[log]}$ and sparse Turing-complete sets for NP, Proc. 2nd IEEE Structure in Complexity Theory Conference (1987), pp. 33–40.

[23] C. KINTALA AND P. FISHER, Refining nondeterminism in relativized complexity classes, *SIAM Journal on Computing* **13** (1984), pp. 129–337.

[24] J. KÖBLER, personal communication, Ulm 1991.

[25] J. KÖBLER, U. SCHÖNING AND J. TORÁN, On counting and approximation, *Acta Informatica* **26** (1989), pp. 363–379.

[26] J. KÖBLER, U. SCHÖNING AND J. TORÁN, The Graph Isomorphism problems is low for PP. Proc. 9th STACS, *Lecture Notes in Computer Science* **577** (Springer, Berlin, 1992), pp. 401–415. To appear in *Journal of Computational Complexity.*

[27] J. KÖBLER, U. SCHÖNING AND J. TORÁN, The Graph Isomorphism problem, its structural complexity, Birkhauser (1993).

[28] M.W. KRENTEL, The complexity of optimization problems, *J. Comput. System Sci.* **36** (1988), pp. 490–509.

[29] R.E. LADNER, The circuit value problem is log space complete for P. *SIGACT NEWS* **7**, pp. 18–20.

[30] R.E. LADNER, N. LYNCH, Relativization of questions about log space computability, *Mathematical Systems Theory,* **10** (1976), pp. 19–32.

[31] R.E. LADNER, N. LYNCH AND A. SELMAN, Comparison of polynomial-time reducibilities, *Theoretical Computer Science* **1** (1975), pp. 103–123.

[32] M. LI, P. VITANYI, "Applications of Kolmogorov complexity in the theory of computation", in *Complexity Theory Retrospective,* A. Selman, ed., Springer-Verlag (1990) pp. 147–203.

[33] T. LOZANO, On parallel versus logarithmic many queries to NP, Technical Report LSI-7-93, Universitat Politècnica de Catalunya (1993).

[34] A. LOZANO AND J. TORÁN, On the non-uniform complexity of the graph isomorphism problem. Proc. 7nd IEEE Structure in Complexity Theory Conference (1992), pp. 118–131.

[35] R. MATHON, A note on the graph isomorphism counting problem. *Information Processing Letters* **8** (1979), pp. 131–132.

[36] A. NAIK, M. OGIWARA, AND A. SELMAN P-selective sets, andreducing search to decision vs. self-reducibility. In this Proceedings.

[37] C. PAPADIMITRIOU, M. YANAKAKIS On limited nondeterminism and the complexity of the VC-dimension. In this Proceedings.

[38] V. PRATT, Every prime has a succint certificate, *SIAM Journal on Computing* **4** (1975), pp. 214–220.

[39] U. SCHÖNING AND K. WAGNER, Collapsing oracle hierarchies, census functions and logarithmically many queries, Proc. 5th. STACS *Lecture Notes in Computer Science* **294** (Springer, Berlin, 1988), pp. 91–98.

[40] A.L. SELMAN, A Taxonomy of Complexity Classes of Functions, Technical Report, State University of New York at Buffalo, Department of Computer Science, Buffalo, NY 14260, November 1991. To appear in *J. Comput. Systems Sci.*

[41] ALAN L. SELMAN, Complexity Classes for Partial Functions, *Bulletin of the EATCS* **45** (1991), pp. 114–130.

[42] A. SELMAN, X. MEI-RUI, AND R. BOOK, Positive relativizations of complexity classes, *SIAM J. Comput.* **12** (1983), pp. 565–579.

[43] S. TODA, On polynomial time truth-table reducibilities of intractable sets to P-selective sets, *Mathematical Systems Theory* **24** 2 (1991), pp. 69–82.

[44] VALIANT, The complexity of computing the permanent. *Theoretical Computer Science*, **8** (1979), pp. 189–201.

[45] K. W. WAGNER, Bounded Query Classes, *SIAM J. Comput.* **19**,5 (1990), pp. 833–846.

[46] O. WATANABE AND S. TODA, Structural Analysis on the Complexity of Inverting Functions, Proc Int. Symposium SIGAL'90 (1990) pp. 31–38.

[47] C. WILSON, Decomposing AC and NC, *SIAM Journal on Comput.* **19**, 2 (1990) pp. 384–396.

Departament de Llengatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

List of research reports (1993).

LSI–93–1–R "A methodology for semantically enriching interoperable databases", Malú Castellanos.

LSI–93–2–R "Extraction of data dependencies", Malú Castellanos and Fèlix Saltor.

LSI–93–3–R "The use of visibility coherence for radiosity computation", X. Pueyo.

LSI–93–4–R "An integral geometry based method for fast form-factor computation", Mateu Sbert.

LSI–93–5–R "Temporal coherence in progressive radiosity", D. Tost and X. Pueyo.

LSI–93–6–R "Multilevel use of coherence for complex radiosity environments", Josep Vilaplana and Xavier Pueyo.

LSI–93–7–R "A characterization of $PF^{NP\|} = PF^{NP[log]}$", Antoni Lozano.

LSI–93–8–R "Computing functions with parallel queries to NP", Birgit Jenner and Jacobo Torán.

Internal reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics (U.P.C.)
Pau Gargallo 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es