# Computing Functions with Parallel Queries to NP [*]

Birgit Jenner [†]
Fakultät für Informatik
Technische Universität München
80290 München, Germany

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

## Abstract

The class $\Theta_2^p$ of languages polynomial-time truth-table reducible to sets in NP has a wide range of different characterizations. We consider several functional versions of $\Theta_2^p$ based on these characterizations. We show that in this way the three function classes $\mathrm{FL}_{\mathrm{log}}^{\mathrm{NP}}$, $\mathrm{FP}_{\mathrm{log}}^{\mathrm{NP}}$, and $\mathrm{FP}_{||}^{\mathrm{NP}}$ are obtained. In contrast to the language case the function classes seem to all be different. We give evidence in support of this fact by showing that $\mathrm{FL}_{\mathrm{log}}^{\mathrm{NP}}$ coincides with any of the other classes then L = P, and that the equality of the classes $\mathrm{FP}_{\mathrm{log}}^{\mathrm{NP}}$ and $\mathrm{FP}_{||}^{\mathrm{NP}}$ would imply that the number of nondeterministic bits needed for the computation of any problem in NP can be reduced by a polylogarithmic factor, and that the problem can be computed deterministically with a sub-exponential time bound of order $2^{n^{O(1/\log\log n)}}$.

# 1 Introduction

The study of nondeterministic computation 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 decision version of it, transforming the problem into a set and then studying the complexity of the set instead. Information about

[*] An extended abstract of this paper was presented at the 8th Structures in Complexity Conference. 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

the complexity of the set is then used to derive information about the complexity of the function. This is not entirely satisfactory since many computational problems are "functional" in nature, and they are not as interesting when considered as decision problems; for example, in general it seems more useful to find a Hamiltonian tour in a graph than to decide whether the graph has such a 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 ourselves to the polynomial-time context, the following three approaches can be distinguished:

- The nondeterministic machine is restricted to output only one value for a given input (on possibly many paths).

For polynomial time this generates the class of (partial) functions NPSV [36]. 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 in polynomial time.

- 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 [40], optP [26], and spanP [24] 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 [26] 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 functional analogue to the class $\Delta_2^{\mathrm{p}}$ of sets Turing reducible to a set in 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 $\Theta_2^p$ [42], the non-adaptive analogue to the adaptive class $\Delta_2^p$. $\Theta_2^p$ is a very robust class that can be characterized in a wide variety of ways [2, 8, 19, 42]. In Section 2 we go over the characterizations obtained for the language class $\Theta_2^{\mathrm{p}}$ observing 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 $\mathrm{FL}_{\log}^{\mathrm{NP}}$, and give some natural examples of functions in these classes.

In contrast to the language case these three function classes seem to be different. In Section 3 we give evidence in support of this fact showing that any of the equalities has unlikely consequences. We prove that $\mathrm{FL}^{\mathrm{NP}}_{\log}$ coincides with any of the other two classes then L = P.

The question whether the classes $\mathrm{FP}^{\mathrm{NP}}_{||}$ and $\mathrm{FP}^{\mathrm{NP}}_{\log}$ are equal has attracted the attention of different researchers. It is known that the hypothesis $\mathrm{FP}^{\mathrm{NP}}_{||} = \mathrm{FP}^{\mathrm{NP}}_{\log}$ implies that FewP=P, NP=R and coNP=US, [4][39][35]. These three consequences follow in fact from a weaker hypothesis, namely from the existence of a polynomial-time algorithm that decides correctly the satisfiability of a formula with at most one satisfying assignment. (If the formula has more than one assignment the algorithm may incorrectly decide that the formula is not satisfiable). More formally (see [15] for the definition of promise problem), if the promise problem (1SAT, SAT) has a solution in P then FewP=P, NP=R and coNP=US.

We present a different consequence of the equality of the function classes $\mathrm{FP}^{\mathrm{NP}}_{||}$ and $\mathrm{FP}^{\mathrm{NP}}_{\log}$ that contrary to the previous results does not seem to be related with the promise problem (1SAT, SAT). We show that if $\mathrm{FP}^{\mathrm{NP}}_{||} = \mathrm{FP}^{\mathrm{NP}}_{\log}$ then there is a reduction by a polylogarithmic factor in the number of nondeterministic bits needed to decide a problem in NP. From this it follows that a polylogarithmic amount of nondeterminism can be simulated in polynomial time, and also that SAT can be decided (for any $k$) in polynomial time with only $O(\frac{n}{\log^k n})$ nondeterministic bits. For the deterministic complexity of the satisfiability problem, we show that in the case $\mathrm{FP}^{\mathrm{NP}}_{||} = \mathrm{FP}^{\mathrm{NP}}_{\log}$ there is a deterministic subexponential time algorithm deciding SAT that works in time $2^{n^{O(1/\log\log n)}}$. This still does not show that the hypothesis implies P = NP but in some sense makes the gap between the classes "smaller". The methods used to prove these results are new; they are based on a combinatorial technique that uses a polynomial-time approximation algorithm of ratio $O(\log n)$ for the Set Cover Problem given by Johnson in [20].

For simplicity all through this article the expression "$\log n$" denotes $\lceil \log_2 n \rceil$ and "$\frac{a}{b}$" denotes integer division of $a$ by $b$.

## 2 Function classes related to $\Theta^{\mathrm{p}}_2$

As mentioned in the introduction, the closure of NP under polynomial-time Turing reducibility defines the language class $\mathrm{P}^{\mathrm{NP}}$ or $\Delta^{\mathrm{p}}_2$ (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 and their related closure classes such as

- $\mathrm{P}^{\mathrm{NP}}_{||}$, the closure of NP under polynomial-time truth-table reducibility or, equivalently, non-adaptive (parallel) reducibility [29].

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.

- $\mathrm{P}^{\mathrm{NP}}_{\log}$, the closure of NP under polynomial-time Turing reducibility with logarithmically many queries [26].

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

- $\mathrm{L}^{\mathrm{NP}}$, the closure of NP under logspace Turing reducibility or, equivalently, $\mathrm{L}_{||}^{\mathrm{NP}}$, the closure of NP under logspace truth-table reducibility or non-adaptive (parallel) reducibility [28].

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 on 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 [3])).

- $\mathrm{L}_{\log}^{\mathrm{NP}}$, the closure of NP under logspace Turing reducibility with logarithmically many queries [42].
- $\mathrm{AC}_0^{\mathrm{NP}}$, the closure of NP under reducibility with logspace-uniform circuits of constant depth [12, 43].

A language $L$ is contained in $\mathrm{AC}_0^{\mathrm{NP}}$, if there is a 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 logspace bounded transducer which on input $1^n$ 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 $\Theta_2^{\mathrm{p}}$ coined by Wagner [42] as the non-adaptive analogue of $\Delta_2^{\mathrm{p}}$ in the Polynomial Hierarchy.

$\Theta_2^p$ is an extremely robust class that still can be characterized in many other ways (see [2, 8, 19, 42]). In this section we study the mentioned characterizations of $\Theta_2^p$ adapting them to compute functions instead of languages. We show that these characterizations generate the three function classes $\mathrm{FP}_{||}^{\mathrm{NP}}$, $\mathrm{FP}_{\log}^{\mathrm{NP}}$ and $\mathrm{FL}_{\log}^{\mathrm{NP}}$. We have selected from the broad list of ways to define $\Theta_2^{\mathrm{p}}$ the above ones since they are particularly interesting in order to illustrate the different concepts involved in the characterizations, like restricted oracle mechanisms, logspace or circuit complexity. Moreover all the characterizations of $\Theta_2^{\mathrm{p}}$ given in [2, 8, 19, 42] that 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 $\Theta_2^{\mathrm{p}}$ that we use later to define function classes.

**Theorem 2.1** [8] [10] [28] [42] *The class* $\Theta_2^{\mathrm{p}}$ *of languages truth-table reducible to* NP *can be characterized as* $\Theta_2^{\mathrm{p}} = \mathrm{P}_{||}^{\mathrm{NP}} = \mathrm{L}_{||}^{\mathrm{NP}} = \mathrm{L}^{\mathrm{NP}} = \mathrm{AC}_0^{\mathrm{NP}} = \mathrm{P}_{\log}^{\mathrm{NP}} = \mathrm{L}_{\log}^{\mathrm{NP}}$.

For the proof of Theorem 2.1, the "census technique" is of particular importance. This technique was developed by Hemachandra [18] for the proof of $\mathrm{P}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{\log}^{\mathrm{NP}}$ [18], and was also successfully used by Kadin in [21]. For a set $L$ in $\mathrm{P}_{||}^{\mathrm{NP}}$ 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 logarithmically 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 and 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.

We show now that the characterizations of the class $\Theta_2^{\mathrm{P}}$ 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 $\mathrm{P}_{||}^{\mathrm{NP}}$, $\mathrm{L}_{||}^{\mathrm{NP}}$, $\mathrm{L}^{\mathrm{NP}}$, and $\mathrm{AC}_0^{\mathrm{NP}}$, the resulting function classes coincide.

**Theorem 2.2** $\mathrm{FL}_{\log}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}} \subseteq \mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FL}_{||}^{\mathrm{NP}} = \mathrm{FL}^{\mathrm{NP}} = \mathrm{FAC}_0^{\mathrm{NP}}$.

**Proof.** (from left to right) The inclusion $\mathrm{FL}_{\log}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}}$ is trivial, because logspace is a restriction of polynomial time. The second inclusion is proved in the same way as in the language case: all possible queries that could be asked (polynomially many) are constructed, and then asked in parallel.

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 $i$th 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 with the computation power of class $\mathcal{C}$ 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 $i$th 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 $\mathrm{L}^{\mathrm{NP}}$. Here $T'$ simply computes one bit $f_i(x)$ after another 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 $\mathrm{L}_{||}^{\mathrm{NP}}$, $T'$ first computes for all $i$ with $1 \leq i \leq p(|x|)$ and $b \in \{0, 1\}$ the sequence of parallel queries that $T$ produces for

$\langle x, i, b\rangle$ and asks all these queries in parallel. Then, with the oracle answers on the query tape $T'$ can compute each $f_i(x)$ deterministically. 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 $y_1 y_2 \ldots y_{2p(n)}$ yields a codification of $f(x)$ in which "10" codifies 0, "01" codifies 1, and "00" codifies a padding symbol. □

All three function classes $\mathrm{FP}_{||}^{\mathrm{NP}}$, $\mathrm{FP}_{\log}^{\mathrm{NP}}$, and $\mathrm{FL}_{\log}^{\mathrm{NP}}$ contain important natural functions. 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 [26]. 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 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 $\mathit{Assign}(F)$ formed by the strings representing satisfying assignments for $F$ plus the string $0^n$. We define the function $\mathit{Sup}$ : {Boolean formulas} $\longrightarrow \{0,1\}^*$. $\mathit{Sup}(F)$ is the supremum of $\mathit{Assign}(F)$ 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 $i \leq n$, if $x$ has a "1" in this position then so does $y$). It is not hard to see that the function $\mathit{Sup}$ is metric complete for the class $\mathrm{FP}_{||}^{\mathrm{NP}}$[23]. More examples of natural complete functions in $\mathrm{FP}_{||}^{\mathrm{NP}}$ have been obtained in [13] and [7].

Another example of a function in $\mathrm{FP}_{||}^{\mathrm{NP}}$ is the one computing the number of isomorphisms between two given graphs [32, 25]. 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 cannot even be computed in polynomial time with access to any class in the polynomial hierarchy (unless this hierarchy collapses).

For a few NP decision problems search functions related to them (functions that for any string in the NP set provide a proof or witness of this fact) can be computed in $\mathrm{FP}_{||}^{\mathrm{NP}}$. This is the case for the Primality and Graph Automorphism problems. The first one is known to be in UP [16] and it can be easily seen that all the problems in UP have search functions in $\mathrm{FP}_{||}^{\mathrm{NP}}$. For the case of Graph Automorphism the method to obtain solutions (automorphisms) in $\mathrm{FP}_{||}^{\mathrm{NP}}$ uses some group theoretic arguments particular to this problem [31, 25]. 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 $\mathrm{FP}_{||}^{\mathrm{NP}}$.

All the given examples are functions in $\mathrm{FP}_{||}^{\mathrm{NP}}$ that are not known to be in its subclass $\mathrm{FP}_{\log}^{\mathrm{NP}}$. Krentel [26] showed that many optimization problems whose solution is polynomially bounded are metric complete for $\mathrm{FP}_{\log}^{\mathrm{NP}}$. Examples of such problems are the function that computes the maximum size of a clique in a graph, and 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 $\mathit{Sup}$ presented above, is the function $\mathit{Sup}'$ that for a Boolean formula $F$, computes the number of "1's" in $\mathit{Sup}(F)$. The function *Sup'* is metric complete for the class $\mathrm{FP}^{\mathrm{NP}}_{\log}$.

All these examples of complete functions in $\mathrm{FP}^{\mathrm{NP}}_{\log}$ belong also to the class $\mathrm{FL}^{\mathrm{NP}}_{\log}$. It does not make sense to talk about polynomial-time metric completeness for this last class since the closure of $\mathrm{FL}^{\mathrm{NP}}_{\log}$ under polynomial-time metric reducibility coincides with $\mathrm{FP}^{\mathrm{NP}}_{\log}$. However, as we will see in the next section $\mathrm{FL}^{\mathrm{NP}}_{\log}$ seems to be a much weaker class than $\mathrm{FP}^{\mathrm{NP}}_{\log}$. In particular "hard" functions in FP (like the Circuit Value function [27]) probably do not belong to $\mathrm{FL}^{\mathrm{NP}}_{\log}$.

The three function classes $\mathrm{FP}^{\mathrm{NP}}_{||}$, $\mathrm{FP}^{\mathrm{NP}}_{\log}$, and $\mathrm{FL}^{\mathrm{NP}}_{\log}$ seem to be all different, since as we will show in Section 3, 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 logarithmically bounded in length, i.e., functions $f$ for which $|f(x)| \in O(\log|x|)$. For a function class $\mathcal{F}$, we denote by $\mathcal{F}[\log n]$ the subclass of $\mathcal{F}$ formed by such functions.

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

**Proof.** Because of Theorem 2.2, it suffices to show $\mathrm{FP}^{\mathrm{NP}}_{||}[\log n] \subseteq \mathrm{FL}^{\mathrm{NP}}_{\log}[\log n]$. 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 $x$). This uses $O(\log n)$ 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 $O(\log n)$. □

Another way to avoid the communication problem between the oracle and the base machine is to allow functions that can produce a string instead of just giving a 0-1 answer as oracles. Now the communication problem mentioned above is avoided because the oracle provides sufficient bits. An approach for this is to use *witness oracles* [9] or NP *multivalued functions* [36, 17]. The function classes defined with the use of these kinds of functional oracles and with the different types of oracle access considered above coincide.

An interesting subclass of $\mathrm{FP}^{\mathrm{NP}}_{||}$is the class NPSV[36] of single-valued NP functions. NPSV is the class of functions for which a single-valued NP transducer computing them exists. Note that this transducer may not produce any output, but if it does, then the output must be the same on all paths.

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 3. 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 $\mathrm{NPSV}^{\mathrm{NP}}$ the class of functions that are computable by an NPSV transducer with restricted access to an oracle in NP, and for a function $f$, $\mathrm{NPSV}_f^{\mathrm{NP}}$ 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 NPSV 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. Hence, it holds $\mathrm{FP}^{\mathrm{NP}} = \mathrm{NPSV}^{\mathrm{NP}}$. But we furthermore obtain the following characterization of $\mathrm{FP}_{||}^{\mathrm{NP}}$.

**Theorem 2.4** $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{NPSV}_{\log}^{\mathrm{NP}}$.

**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 using $O(\log n)$ 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 an 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 and therefore 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. □

It is not hard to see that Theorem 2.4 can be generalized. It holds for any $k \geq 0$, $\mathrm{FAC}_k(\mathrm{NP}) = \mathrm{NPSV}_{\log^{k+1}}^{\mathrm{NP}}$, where $\mathrm{FAC}_k(\mathrm{NP})$ denotes the class of functions computable by logspace uniform unbounded fan-in circuits of depth $O(\log^k)$ and polynomial size ([43]).

# 3 Consequences of the equality of the function classes

In this section we present evidence 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 3.1** $\mathrm{FP}_{\log}^{\mathrm{NP}} = \mathrm{FL}_{\log}^{\mathrm{NP}}$ *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* $\in$ 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 logspace 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 [27]) can be computed in logspace as follows. On input of $C$ and $x$, a logspace 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 $f \in \mathrm{FP}^{\mathrm{NP}}_{\log}$. Let $c \log n$ be the bound on the number of queries for an input $x$ of length $n$. First, note that only logspace and at most $2c \log n$ queries to the following oracle $A'' \in \mathrm{NP}$ are neccessary to obtain the correct query answer sequence of $T$ on input $x$.

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

Now, define the function $f''$ with

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

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

From the above proof it also follows that even the hypothesis $\mathrm{FP} \subseteq \mathrm{FL}^{\mathrm{NP}}_{\log}$ would imply $\mathrm{L} = \mathrm{P}$. Also as a consequence of Theorem 3.1, it seems unlikely that parallel queries to NP can be reduced to logarithmic adaptive queries with logspace.

**Corollary 3.2** *If* $\mathrm{FL}^{\mathrm{NP}}_{||} \subseteq \mathrm{FL}^{\mathrm{NP}}_{\log}$, *then* $\mathrm{L} = \mathrm{P}$.

The equality of the classes $\mathrm{FP}^{\mathrm{NP}}_{||}$ and $\mathrm{FP}^{\mathrm{NP}}_{\log}$ would also imply strong consequences; this question has been studied by different reseachers. From the results of Toda in [39] it can be concluded that if the classes $\mathrm{FP}^{\mathrm{NP}}_{||}$ and $\mathrm{FP}^{\mathrm{NP}}_{\log}$ coincide then there is a polynomial-time algorithm that decides correctly the satisfiability of a formula with at most one satisfying assignment. (If the formula has more than one assignment the algorithm may incorrectly decide that the formula is not satisfiable). This result is stated formally using the concept of promise problems (see [15]). 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 3.3** [4, 39] *If* $\mathrm{FP}^{\mathrm{NP}}_{||} \subseteq \mathrm{FP}^{\mathrm{NP}}_{\log}$, *then the promise problem* $(\mathrm{1SAT}, \mathrm{SAT})$ *has a solution in* P.

A polynomial-time solution for the promise problem (1SAT, SAT) would imply the unexpected consequences expressed in the next theorem. The complexity classes FewP and R mentioned in the result are well-known and we refer to the standard literature for definitions. US is the class of languages computed by a polynomial-time nondeterministic Turing machine that accepts an input if it produces exactly one accepting path [6].

**Theorem 3.4** *If the promise problem* (1SAT, SAT) *has a solution in* P *then* FewP=P, NP=R *and* coNP=US.

The first equality is due to [39], and the second one is due to [41].To our knowledge the third equality is new (and its proof is left as an easy exercise to the reader). The following corollary summarizes these results; with the exception of the observation about co-NP and US, the result appears in this form in [35].

**Corollary 3.5** *If* $\mathrm{FP}_{||}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}}$, *then* FewP=P, NP=R *and* coNP=US.

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 $\frac{n}{\log^k n}$ nondeterministic bits.

To obtain these results we use the fact that the equality of the function classes $\mathrm{FP}_{||}^{\mathrm{NP}}$ and $\mathrm{FP}_{\log}^{\mathrm{NP}}$ can be characterized in a very useful way with the concept of polynomial enumerators. 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 question $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ is equivalent to whether every function in $\mathrm{FP}_{||}^{\mathrm{NP}}$ has a polynomial enumerator. (The implication from left to right is easy; for the other implication, if a function $f \in \mathrm{FP}_{||}^{\mathrm{NP}}$ has a polynomial enumerator, the value of $f(x)$ can be identified querying NP to obtain first the census $k$ of queries answered positively in the computation of $f(x)$, and then doing a binary search over the set of indices of the values produced by the enumerator to obtain the unique one that could be computed with $k$ correct positive oracle answers (see [30]).)

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 $\frac{n}{\log^k n}$ nondeterministic bits.

To prove our results we state the following theorem:

**Theorem 3.6** *If* $\mathrm{FP}_{||}^{\mathrm{NP}} \subseteq \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then there is a function* $f \in \mathrm{NPSV}_{\log\log}^{\mathrm{NP}}$ *that for a sequence of Boolean formulas* $F_1, \ldots, F_n$ *outputs one satisfiable formula from the list in case one exists. (If all the formulas are unsatisfiable then the value of* $f$ *is some special symbol).*

**Proof.** 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 there is a polynomial enumerator for $g$. Running the enumerator on $F_1, \ldots, F_n$ one can obtain a list $L = (v_1, \ldots, v_{p(n)})$ of polynomially many potential values for $g$, 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 $V = \{1, \ldots, p(n)\}$ and $E = \{(i, j) \mid v_j \prec v_i\}$ ($v_j \prec v_i$ 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 $v_i$). For every $i \leq p(n)$, we say that the string $v_i$ is the value of node $i$.

We contract $G$ to obtain a simpler graph $G_1$ using the following contraction rules in the written order.

- Delete the nodes with value $0^n$.
- If $i$ is a node without descendants (a leaf) and its value, $v_i$, has more than one "1" then turn to "0" all the "1's" in $v_i$ except the first one.
- For every node $i$ assign $v_i$ to the supremum of the values of the leaves that can be reached from $i$.
- Contract all the nodes that have the same value into a single node.

We define the level of a node $i$ as the number of "1's" from its value $v_i$. Also we will say that a node $i$ is consistent with the characteristic vector of $F_1, \ldots, F_n$ (or just that $i$ is consistent) if $v_i$ is smaller than or equal to this value (in the lattice order). 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:

1. If $i$ is a node of level $k$ in $G_1$ then $i$ reaches exactly $k$ leaves.
2. 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 $k$ (the value of this node is the supremum of the consistent leaves in $G_1$).
3. In $G_1$ there is at least one node consistent with the characteristic vector of $F_1, \ldots, F_n$.

Property 1 is straightforward since by the way the graph $G_1$ is constructed, the "1's" in the value of a node correspond to the leaves it reaches. 2 follows from the fact that the supremum of the consistent leaves in $G_1$ is unique, and by construction a node with this value belongs to $G_1$. 3 is true since the value of every node in $G_1$ has at least a "1" and in $G$ at least one of the nodes has a consistent value.

Because of properties 2 and 3, it suffices to obtain the maximum level with a consistent node in $G_1$. Once the level is known, the unique consistent node on 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 3.7** *Let $p$ be a polynomial and $A_p$ be the set of pairs $(n, G)$ where $n \in \mathbb{N}$ and $G$ is a DAG formed from a set of strings in $\{0,1\}^n$ as explained above and contracted following the given rules, with at most $p(n)$ nodes. There is a polynomial-time algorithm that (for sufficiently large $n$), on input $(n, G) \in A_p$, with $G$ having $m$ leaves, selects a set $S$ of at most $\frac{m}{2}$ leaves 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 $\frac{n}{2}$ leaves. We construct a new graph $G_2$ by turning to "0" the positions of the nodes in $G_1$ corresponding to the leaves 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 keep only the remaining formulas.

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 3 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 $\frac{n}{2}$ leaves 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 $G_r$ ($r \leq \log n$) with no node at a level $k \geq \log^4 n$, is obtained.

Given the collection of graphs $G_1, \ldots, G_r$, 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 uniquely determined in a nondeterministic way. Each "1" in the value of 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 value. It follows that the claimed function $f$ is in $\mathrm{NPSV}^{\mathrm{NP}}_{\log\log}$.

We give now the proof of the claim.

**Proof of the claim.** Let $p$ be a polynomial, $n \in \mathbb{N}$ and let $G$ be a graph with $m$ leaves ($m \geq \log^4 n$) and $(n, G) \in A_p$. 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 every combination $C$ of $m - \frac{m}{\log^2 n}$ leaves there must be a node $i$ at a level $k \geq \log^4 n$ such that all the leaves that can be reached from $i$ belong to $C$. We say in this case that $i$ is a *bad* node for $C$. Each node $i$ can be *bad* for at most

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

combinations of $m - \frac{m}{\log^2 n}$ leaves. This is because $i$ has at least $\log^4(n)$ leaf descendants and all these leaves must be in the combination. There are $\binom{m}{m - \frac{m}{\log^2 n}}$ combinations of $m - \frac{m}{\log^2 n}$ leaves. If $b$ is the number of bad 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

$$\begin{aligned} b &\geq \frac{m(m-1)\ldots(m-\log^4 n+1)}{(m - \frac{m}{\log^2 n})(m - \frac{m}{\log^2 n} - 1)\ldots(m - \frac{m}{\log^2 n} - \log^4 n + 1)} > \\ &> \left(\frac{m}{m - \frac{m}{\log^2 n}}\right)^{\log^4 n} > \left(1 + \frac{1}{\log^2 n}\right)^{\log^4 n} > 2^{\log^2 n}, \end{aligned}$$

which is a contradiction (for sufficiently large $n$) since $b \leq p(n)$.

We have proven the existence of a set of leaves of size $\frac{m}{\log^2 n}$ that "covers" each node at level $k \geq \log^4 n$. However the problem to find such a set of leaves of minimum size, is an instance of the search version of the Set Cover problem which is NP-hard. Fortunately, it has been shown by Johnson [20] that the greedy algorithm obtains an approximated solution for the Set Cover 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 $\frac{m}{\log^2 n}$ leaves covering all the nodes at levels higher than or equal to $\log^4 n$, the greedy algorithim obtains in polynomial time and for some constant $c$ a cover of size $\frac{cm}{\log n} < \frac{m}{2}$ (for sufficiently large $n$). □

An interesting observation is that the satisfiable formula selected from a sequence $F_1, \ldots, F_n$ by the function $f$ in the proof is not necessarily the first satisfiable one in the sequence, the selection depends on the enumerator for the function $g$.

The following lemma based on the above result shows that from the hypothesis $\mathrm{FP}^{\mathrm{NP}}_{||} \subseteq \mathrm{FP}^{\mathrm{NP}}_{\log}$ follows that a polylogarithmic number of bits of a satisfiable assignment of a formula can be obtained with parallel queries to NP.

**Lemma 3.8** *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.** We use the characterization $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{NPSV}_{\log}^{\mathrm{NP}}$. 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 obtained formulas, we get a list of Boolean formulas $F_1, \ldots, F_{p(n)}$. By the same argument as in Theorem 3.6, a satisfiable formula $F_i$ (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 $\log\log n$ queries determine a level in the constructed graph, and the level uniquely determines the formula $F_i$. 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 ($\log\log n$ many bits encoding a level in the reduced graph) 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 queries 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 $\frac{\log^{k+1} n}{(\log\log n)^k}$. The NP oracle has to be told how often to iterate, i.e. part of the input to the NP oracle is a field $j$ indicating to answer queries about the $j$-th iteration.

After all the queries are made, a single valued nondeterministic computation outputs the obtained partial assignment. □

Based on these results we obtain some consequences that make reference to subclasses of NP with bounded nondeterminism. Introduced in [22], these subclasses have also been considered in [14] and [34]. For a polynomially bounded 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 $A \in \mathrm{P}$ and a constant $c \in \mathbb{N}$ such that for every string $x$

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

The next theorem shows that under the hypothesis $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$, a significant reduction in the number of nondeterministic bits in an NP computation can be obtained.

**Theorem 3.9** *If* $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then for any polynomial-time computable and polynomially bounded function* $f : \mathbb{N} \longrightarrow \mathbb{N}$ *and for any* $k \in \mathbb{N}$, $\mathrm{NP}(f) \subseteq \mathrm{NP}(\frac{f}{\log^k})$.

**Proof.** Let $f$ be a polynomially bounded function, $k \in \mathbb{N}$ and $L$ be a set in $\mathrm{NP}(f)$. There is a polynomial-time predicate $A$ and a constant $c$ such that for every string $x$

$$x \in L \Longleftrightarrow \exists y,\ |y| \leq cf(|x|) \wedge \langle x, y\rangle \in A.$$

Let $\langle x, z\rangle$ be a pair of strings such that $x \in L$ and $z$ is the prefix of some string $w$ with $|w| \leq cf(|x|)$ and $\langle x, w\rangle \in A$. Then by (the proof of) Lemma 3.8 there is a function

$g \in \mathrm{FP}_{||}^{\mathrm{NP}}$ that on input $\langle x, z\rangle$ produces a sequence $z'$ of length $\log^{k+1}(|x|)$ such that $zz'$ can be extended to a string $y$ ($|y| \leq cf(|x|)$) with $\langle x, y\rangle \in A$. By the hypothesis $g \in \mathrm{FP}_{\log}^{\mathrm{NP}}$; let $M$ be the machine computing $g$ with the help of $\log n$ queries to NP. We can construct a nondeterministic algorithm to compute $L$ using only $O(\frac{f}{\log^k})$ nondeterministic bits in the following way. On input $x$ the algorithm simulates $M(\langle x, \lambda\rangle)$ but instead of querying the oracle, it guesses a possible answer for each query. In this way it produces a string $z$ of length $\log^{k+1}(|x|)$ guessing $O(\log(|x|))$ bits. The algorithm simulates again $M$ this time on input $\langle x, z\rangle$, and repeats the process $\frac{cf(|x|)}{\log^{k+1}(|x|)}$ times obtaining a string $y$ of length $cf(|x|)$. If for some prefix $y'$ of $y$ it holds $\langle x, y\rangle \in A$ the algorithm accepts, otherwise it rejects. (The algorithm has to check the prefixes of $y$ since the witness for $x$ can be smaller than $cf(|x|)$.) It should be clear that the algorithm works in polynomial time, decides $L$ correctly, and the number of nondeterministic bits it needs is in $O(\frac{f}{\log^k})$. □

The next two results are straigtforward consequences of Theorem 3.9.

**Corollary 3.10** *If* $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then for any* $k \in \mathbb{N}$*, the class* $\mathrm{NP}(\log^k)$ *is included in* P*.*

**Corollary 3.11** *If* $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then for any* $k \in \mathbb{N}$, $\mathrm{SAT} \in \mathrm{NP}(\frac{n}{\log^k n})$.

The next result shows that if the equality of the function classes $\mathrm{FP}_{||}^{\mathrm{NP}}$ and $\mathrm{FP}_{\log}^{\mathrm{NP}}$ holds then every problem in NP can be computed in deterministic subexponential time $2^{n^{O(1/\log\log n)}}$. This result was pointed out to us by O. Watanabe. Its proof follows the lines of Claim 3 in [38]. For completeness we give a proof of this result.

**Corollary 3.12** *If* $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ *then* $\mathrm{NP} \subseteq \mathrm{DTIME}(2^{n^{O(1/\log\log n)}})$.

**Proof.** Consider the universal set

$$\begin{aligned} \mathit{UNIV} = \ \{ \ & \langle M, x, 0^d, 0^t\rangle \mid \text{there is a string } w,\ |w| \leq d \text{ such that the deterministic} \\ & \text{machine } M \text{ accepts input } \langle x, w\rangle \text{ in at most } t \text{ steps}\}. \end{aligned}$$

We show first that there is a deterministic machine $M'$ and a polynomial $p$ that for every input $\tau = \langle M, x, 0^d, 0^t\rangle$ satisfies

$$\tau \in \mathit{UNIV} \iff \langle M', \tau, 0^{d'}, 0^{t'}\rangle \in \mathit{UNIV},$$

where $d' = \frac{d}{\log(|x|)}$ and $t' = p(|\tau|)$.

The proof of this fact is similar to the one for Theorem 3.9. By (the proof of) Lemma 3.8 there is a function $f \in \mathrm{FP}_{||}^{\mathrm{NP}}$ that for an input $\langle M, x, 0^d, 0^t\rangle$ in *UNIV* produces a string $w$ of length $\log^2(|x|)$ such that $w$ can be extented to a string $y$ of length $d$ and $M$ accepts $\langle x, y\rangle$ in at most $t$ steps. By the hypothesis $f \in \mathrm{FP}_{\log}^{\mathrm{NP}}$. There is an algorithm $N$ that simulates the algorithm computing $f$ but guessing nondeterministically the oracle answers, and repeats the process $\frac{d}{\log^2(|x|)}$ times as in Theorem 3.9 to obtain all $d$ bits from $y$. The machine $M'$ we are looking for on input $\langle \tau, w\rangle$, with $|w| = d'$, just checks that $N$ with the sequence $w$ of nondeterministic bits produces a string $y$ such that $\langle x, y\rangle$ is accepted by $M$ in at most $t$ steps. Observe that the running time of $M'$ is bounded by a polynomial that depends only on the running time of the algorithm computing the function $f$.

The problem $\langle M, x, 0^d, 0^t\rangle \in$ *UNIV* has been reduced to the problem $\langle M', \tau, 0^{d'}, 0^{t'}\rangle \in$ *UNIV*. The time bound $t'$ increases, but the amount of nondeterminism decreases by a logarithmic factor.

Let $L$ be a set in NP. There is a nondeterministic machine $M$ and a polynomial $q$ such that for every string $x$, $x \in L \iff \langle M, x, 0^{q(|x|)}, 0^{q(|x|)}\rangle \in$ *UNIV*. Define $x_0 = x$, $d_0 = q(|x|)$, $t_0 = q(|x|)$ and $\tau_0 = \langle M, x, 0^{d_0}, 0^{t_0}\rangle$, and define inductively for $i > 0$ $x_i = \tau_{i-1}$, $d_i = \frac{d_{i-1}}{\log(|x|)}$, $t_i = p(|\tau_{i-1}|)$ and $\tau_i = \langle M', x_i, 0^{d_i}, 0^{t_i}\rangle$. Observe that there is a polynomial $r$ such that for each $i$, $|\tau_i| \leq r(t_i)$. Let $m$ be the first integer such that $d_m \leq \log(|x|)$. Since $d$ decreases each time by a logarithmic factor, for some constant $c$, $m \leq \frac{c\log(|x|)}{\log\log(|x|)}$. It also holds that $\tau_0 \in$ *UNIV* $\Leftrightarrow \tau_1 \in$ *UNIV* $\Leftrightarrow \ldots \Leftrightarrow \tau_m \in$ *UNIV*, and therefore to decide whether $x \in L$ it suffices to show whether $\tau_m \in$ *UNIV*. $\tau_m$ is computable deterministically in polynomial time with respect to $|\tau_m|$. Also, deciding whether $\tau_m \in$ *UNIV* can be done in polynomial time with respect to its length. Because of these two facts the complexity of deciding whether $x \in L$ is bounded by a polynomial in $|\tau_m| \leq r(t_m)$. To evaluate $t_m$ observe that

$$t_i = p(|\tau_{i-1}|) \leq p(r(t_{i-1})) \leq \ldots \leq p(r(\ldots(r(p(q(|x|))))\ldots)).$$

Therefore for some constant $k$ (depending on the polynomials $p$, $q$ and $r$),

$$t_m \leq |x|^{\left(k^{\frac{c\log(|x|)}{\log\log(|x|)}}\right)},$$

which gives a bound of order $2^{n^{O(1/\log\log n)}}$ for the complexity of $L$. □

In [38] it is proven that from the hypothesis that every NP set is bounded truth-table reducible to a P-selective set, follows NP $\subseteq$ DTIME$(2^{n^{O(1/\sqrt{\log n})}})$, which is a better bound for NP than the one in Corollary 3.12. This result was improved in [1, 5, 33] to show that under the same hypothesis P=NP. It is not hard to see that the mentioned hypo thesis implies $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ [37], although the converse does not seem to be true. Because of this, the mentioned results in [38, 1, 5, 33] and Corollary 3.12 seem incomparable.

# 4 Conclusions

We have shown that the existing characterizations of the language class $\Theta_2^p$, when adapted to compute functions, generate the classes $\mathrm{FL}_{\log}^{\mathrm{NP}}$, $\mathrm{FP}_{\log}^{\mathrm{NP}}$ and $\mathrm{FP}_{||}^{\mathrm{NP}}$. We have given evidence that these classes are all different showing that $\mathrm{FL}_{\log}^{\mathrm{NP}}$ coincides with any of the other two classes then L = P, and that the equality of the two last classes would imply a polylogarithmic reduction in the number of nondeterministic bits needed to compute a problem in NP. It remains an open question whether the last result can be improved to show that the question $\mathrm{FP}_{||}^{\mathrm{NP}} = \mathrm{FP}_{\log}^{\mathrm{NP}}$ can be completely characterized by the equality of two language complexity classes (like for example P = NP).

# Acknowledgement

We would like to thank J.L. Balcázar, R. Beigel, H. Buhrman, R. Gavaldá, T. Hagerup, J. Köbler, T. Lozano, P.B. Miltersen and C. Wilson for interesting discussions on the topics of the paper and the two anonymous referees for helpful suggestions.

# References

[1] M. Agrawal and V. Arvind, Polynomial time truth-table reductions to p-selective sets. To appear in the proceedings of the 9th Structure in Complexity Theory Conference, 1994.

[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, 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")

[4] R. Beigel, NP-hard sets are p-superterse unless R=NP. Tech. Report TR4, Dept. of Comp. Science, John Hopkins University, 1988.

[5] R. Beigel, M. Kummer and F. Stephan, Approximable sets. To appear in the proceedings of the 9th Structure in Complexity Theory Conference, 1994.

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

[7] H. Buhrman, J. Kadin and T. Thierauf, On functions computable with nonadaptive queries to NP. Manuscript 1994.

[8] 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.

[9] S.R. Buss, J. Krajicek and G. Takeuti, Provably total functions in bounded arithmetic theories $R^i_3$, $U^i_2$ and $V^i_2$. To appear in: Arithmetic, Proof Theory and Computational Complexity. ed: P. Clote and J. Krajicek, Oxford University Press.

[10] J. Castro and C. Seara, Characterizations of some complexity classes between $\Theta^p_2$ and $\Delta^p_2$. 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] Z.Z. Chen and S. Toda, On the complexity of computing the minimum subgraph. Submitted (1990).

[14] J. Díaz and J. Torán, Classes of bounded nondeterminism, *Mathematical Systems Theory,* **23** (1990), pp. 21–32.

[15] 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.

[16] 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.

[17] 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.

[18] L. Hemachandra, The sky is falling: The strong exponential hierarchy collapses, Tech. Report TR86-777, Deptartment of Computer Science, Cornell University (1986).

[19] B. Jenner, B. Kirsig and K.-J. Lange, The logarithmic alternation hierarchy collapses $\mathrm{A}\Sigma_2^{\mathrm{L}} = \mathrm{A}\Pi_2^{\mathrm{L}}$, *Information and Computation* ,1 (1989), pp. 269–288.

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

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

[22] C. Kintala and P. Fisher, Refining nondeterminism in relativized complexity classes, *SIAM Journal on Computing* **13** (1984), pp. 129–337.

[23] J. Köbler, personal communication, Ulm 1991.

[24] J. Köbler, U. Schöning and J. Torán, On counting and approximation, *Acta Informatica* **26** (1989), pp. 363–379.

[25] J. Köbler, U. Schöning and J. Torán, The Graph Isomorphism problem, its structural complexity, Birkhauser (1993).

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

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

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

[29] R.E. Ladner, N. Lynch and A. Selman, Comparison of polynomial-time reducibilities, *Theoretical Computer Science* **1** (1975), pp. 103–123.

[30] T. Lozano, Parallel versus logarithmic many queries to NP, Report LSI-7-93-R, Universitat Politècnica de Catalunya, Barcelona (1993).

[31] 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.

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

[33] M. Ogiwara, Polynomial-time membership comparable sets. To appear in the proceedings of the 9th Structure in Complexity Theory Conference, 1994.

[34] C. Papadimitriou, M. Yannakakis, On limited nondeterminism and the complexity of the VC-dimension. Proc. 8th IEEE Structure in Complexity Theory Conference (1993), pp. 12–19.

[35] 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.*

[36] A. Selman, X. Mei-rui, and R. Book, Positive relativizations of complexity classes, *SIAM J. Comput.* **12** (1983), pp. 565–579.

[37] T. Thierauf, Personal communication, 8th Structure in Complexity Conference (1993).

[38] T. Thierauf, S. Toda and O. Watanabe, On sets bounded truth-table reducible to P-selective sets, Technical report 93TR-0013, Dept. of Computer Science Tokyo Institute of Technology (1993). To appear in STACS 94.

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

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

[41] L. Valiant and V. Vazirani, NP is as easy as detecting unique solutions. *Theoretical Computer Science*, **47** (1986), pp. 85–93.

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

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