Non-Erasing, Counting and Majority over the Linear Time Hierarchy

Arnaud Durand
LACL, Département Informatique,
Université Paris XII, 61 avenue du général de Gaulle,
94010 Créteil Cedex, France
durand@univ-paris12.fr

Malika More
LLAIC, IUT Département Informatique
Université d'Auvergne, Campus des Cézeaux,
BP 86, 63172 Aubière Cedex, France
Malika.More@llaic.u-clermont1.fr

Abstract

In this paper, we investigate several extensions of the Linear Time Hierarchy (denoted by LTH). We first prove that it is not necessary to erase the oracle tape between two successive oracle calls, thereby lifting a common restriction on LTH machines. We also define a natural counting extension of LTH and show that it corresponds to a robust notion of counting bounded arithmetic predicates. Finally, we show that the computational power of the majority operator is equivalent to that of the exact counting operator in both contexts.

1 Introduction

The formalization of linear time is of great significance in computational complexity, because of its practical importance in algorithm design. As far as linear time on Random Access Machines is concerned, Grandjean's introduction of the class $NLIN$ has proved to be very useful both from the point of view of its intrinsic robustness, and because it is large enough to contain most classical NP-complete problems (see [11, 12]). The case of Turing machines is less agreeable to deal with, since both deterministic and nondeterministic linear time are highly dependent on models of computing and input encodings. This is why studies are being carried out into slightly larger and hopefully more robust classes, like for instance the classes defined in [26, 14, 10, 21]. Another interesting approach in the study of feasible complexity classes can be found in [5]. The authors define the so-called "Sharply Bounded Hierarchy" (SBH, for short) which is a hierarchy of classes within P using quasilinear time computation and quantification over strings of logarithmic length. SBH appears to be very natural and has been shown to have a number of both alternative definitions and complete problems. In this paper, we are interested in a linear time analogue of the well-known Polynomial Time Hierarchy on Turing machines (see [28]), namely the Linear Time Hierarchy (see [33]), denoted by LTH. Compared to SBH, more non-determinism is allowed but computation is restricted to linear time. This class is also known to be quite robust, since it can also be defined via alternating Turing machines or stack register machines (see [2]). Moreover, it has several additional characterizations in logic, primitive recursion (via bounded minimum), formal languages, and arithmetic, the latter, namely the rudimentary predicates (see [19, 17, 33]), being very easy to use. In addition, LTH also contains many interesting problems, since it contains $NLIN$[1].

The study of variant machine models is central to the interest of theoretical computer science. In this context, we investigate several extensions of LTH. We first add the notion of linear time oracle

[1] This inclusion holds because $NLIN$ is expressible within the existential fragment of monadic second-order logic with addition (see [13]), whereas LTH corresponds exactly to monadic second-order logic with addition.

non-erasing algorithms to the numerous definitions of LTH (for a review of models of relativization in space-bounded computation, see [6]). Then we study analogues for the Linear Time Hierarchy of certain concepts introduced for the Polynomial Time Hierarchy, namely counting and majority. Counting is a crucial and natural operation in algorithmics and counting classes have been widely studied (see for instance [31, 29, 30, 32]). Moreover, relations (on numbers or graphs) that require counting arguments are among those that have the greatest computational complexity. For instance, Toda's theorem ($PH \subseteq P(\sharp P)$, see [29]) expresses the fact that one single counting oracle is as least as strong as any number of nested ordinary oracles. The main interest of majority is that it leads to formulas that are more natural than formulas obtained by counting. In order to cope with these two properties, we introduce a natural and robust extension of LTH, denoted by $\sharp LTH$, which can be defined either via Turing machines or arithmetics and via both counting and majority.

Section 2 is devoted to the presentation of our model of computation and to the precise definition of LTH.

The first extension of LTH we are interested in is discussed in Section 3. In order to limit the amount of information provided by the oracle in LTH algorithms, it is commonly assumed that before the next query is asked, the previous one is erased (see for instance [33] and [15]). In particular, this implies that the sum of the length of the queries is linear and that the algorithm cannot perform too many long queries. We show that this assumption is superfluous. In other words, with or without this restriction, the defined complexity class remains the same, that is, LTH. What this new model for LTH means is that it is now possible for the machine to re-use the previous query in writing the next one.

The second extension of LTH, denoted by $\sharp LTH$, is presented in Section 4. Its aim is to take into account the possibility of counting in linear time algorithms. This class consists in a rather simple and natural extension of LTH : classical oracles are replaced by oracles that count accepting computations. We prove that this model is equivalent to the class of counting rudimentary sets $\mathcal{R}^\sharp$, previously studied in [25, 9, 8]. The model we propose is not the first time this subject has been attempted. Various types of counting over LTH have been previously considered. For instance, [24, 23] deal with modular counting and group counting over rudimentary relations. These results have been considerably extended in a recent paper [7]. Using stack register machines, Clote shows that for all finite unsolvable groups, the closure of LTH under group counting corresponds to $ALINTIME$. Concerning ordinary counting, it can be shown that $\mathcal{R}^\sharp \subseteq ALINTIME$ (see [3]) but equality is an open problem. Finally, fewness in counting over rudimentary sets is considered in [25, 8]. The complexity approach we propose here is based on classical notions that have been extensively studied over the last few years and takes into account the full power of usual counting over rudimentary relations.

Finally, in Section 5, we consider majority instead of exact counting. We first introduce a majority operation over arithmetic predicates that is satisfied iff more than half of the possible values of a variable satisfy a given predicate. We show that $\mathcal{R}$ is extended by exactly the same amount as with exact counting. Then, a new characterization of $\sharp LTH$ in terms of probabilistic linear time oracle computation is derived.

2 Preliminaries

We consider oracle multitape Turing machines. For such a machine $\mathcal{M}$, we adopt the following notations and conventions. Let $k \geq 2$ be the number of (semi-infinite) tapes of $\mathcal{M}$: tape 1 is the read-only input tape, and tape 2 is a distinguished oracle tape. Let Q be the set of states of $\mathcal{M}$. Five states in Q are distinguished: q_0 (initial state), q_a (accepting state), q_{call} (query, or oracle, state), q_{yes} and q_{no} (answer states). Let us denote by $A = \{1, 2, B\}$ the alphabet[2], and by M the set of possible moves of the heads: `m` (motionless), `r` (right) and `l` (left). The transition table of $\mathcal{M}$ is denoted by δ:

$$\delta : \begin{array}{rl} Q \times A^k & \longrightarrow \mathcal{P}(Q \times (A \times M)^k) \\ (q, a_1, a_2, \ldots, a_k) & \longmapsto \{\ldots, (q', (a_1, m_1), (b_2, m_2), \ldots, (b_k, m_k)), \ldots\}. \end{array}$$

Note that δ is non-deterministic here.

At any step during a computation on input x relative to the oracle set $\mathcal{A}$, the machine $\mathcal{M}$ acts as follows :

[2]The dyadic notation (i.e. words over $\{1, 2\}$) provides a one-to-one and onto correspondence between words and integers, and avoids problems with leading zeros.

- The current state of $\mathcal{M}$ is not its query state q_{call}. Then, the next step in the computation of $\mathcal{M}$ depends only on δ.
- The current state of $\mathcal{M}$ is q_{call}. The next step is then determined by the oracle set $\mathcal{A}$ and the word z written on the oracle tape. We will use the following convention :

  $\delta(q_{call}, a_1, a_2, \ldots, a_k) = (q_{yes}, (a_1, \mathtt{m}), (a_2, \mathtt{m}), \ldots, (a_k, \mathtt{m}))$ if the word z written on tape 2 (the oracle tape) belongs to $\mathcal{A}$ and, if not, $\delta(q_{call}, a_1, a_2, \ldots, a_k) = (q_{no}, (a_1, \mathtt{m}), (a_2, \mathtt{m}), \ldots, (a_k, \mathtt{m}))$.

  Between two successive oracle calls, it is assumed that the *oracle tape is erased* by the call itself.

Finally, let l be the cardinality of the set $Q \cup M \cup A$. In Section 4, we use the l-ary expansion of integers (i.e. each symbol of the Turing machines above is encoded by a specific l-ary digit).

Keeping our computation model in mind, let us now define more precisely the classes we are interested in. We denote by $LinTime$ (resp. $NLinTime$) the class of dyadic languages accepted by a deterministic (resp. non-deterministic) multi-tape Turing machine in linear time. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are two complexity classes, we denote by $\mathcal{C}_1(\mathcal{C}_2)$ the class of languages accepted by some Turing machine in $\mathcal{C}_1$, with an oracle set in $\mathcal{C}_2$.

Definition 2.1 Let LTH (the Linear Time Hierarchy) be the union of the following classes Σ_i^{lin} for $i \geq 0$. $\Sigma_0^{lin} = LinTime$, and for all $i > 0$, $\Sigma_i^{lin} = NLinTime(\Sigma_{i-1}^{lin})$.

Note that it is known that $LogSpace \subseteq LTH \subseteq LinSpace$ (see [22, 20] and [15] pp. 285-287 for a nice proof of $LogSpace \subseteq LTH$). Let us now present the close connection between LTH and bounded arithmetical relations.

Definition 2.2 We denote by $\mathcal{R}$ (for *Rudimentary* relations) the smallest class of relations over integers containing the graphs of addition and multiplication (seen as ternary relations) and closed under Boolean operations ($\neg$, $\wedge$, $\vee$), explicit transformations (i.e. adding, cancelling, renaming, permuting and confusing variables, see a precise definition in [27]), and variable bounded quantifications (i.e. $\forall x < y \ldots$ meaning $\forall x \ (\ x < y \ \rightarrow \ \ldots \ )$ and $\exists x < y \ldots$ meaning $\exists x \ (\ x < y \ \wedge \ \ldots \ )$).

Rudimentary relations (or bounded arithmetical relations) were first introduced by Smullyan in [27]. The following result was proved by Wrathall. Note that the encoding of k-ary relations into languages and conversely (in a rudimentary or linear time way) is classical, such that it does not matter if LTH is a class of languages over some alphabet whereas $\mathcal{R}$ is a class of relations over integers.

Theorem 2.3 ([33]) $LTH = \mathcal{R}$

The intrinsic robustness of the class $\mathcal{R}$ of rudimentary relations is emphasized by the many variants that can be used in its definition. For instance, in [16], Harrow proves that *polynomially* bounded quantifications (such as $\forall x < y^2$) can be made use of instead of variable bounded quantifications. It also appears that $\mathcal{R}$ is a quite large class since most natural relations over integers are rudimentary. For instance, Bennett shows in [4] that the ternary relation $x = y^z$ is rudimentary. Using such powerful tools, it is not difficult to verify that the following two relations (widely used in Section 4) are rudimentary : $y = \textsc{Digit}(l, x, i)$ (y is the digit of rank i of the l-ary expansion of x) and $y = \textsc{Length}(l, x)$ (y is the length of the l-ary expansion of x).

3 Non-erasing Oracle Computation

As far as time-bounded oracle machines are concerned, it is widely assumed that before the next query is asked, the previous one is erased (see a comprehensive survey on this subject in [6]). Here, *we give up this assumption* and introduce an alternative definition of linear time oracle computation. In such a model, the following algorithm becomes valid : write a letter, ask whether it belongs to the oracle, add a letter, ask whether the resulting word belongs to the oracle, and so on. Let us illustrate this strategy of oracle calls.

Consider the following relation : $z = \sharp\{y < x \mid y = 2^n + 1 \text{ and } \textsc{Prime}(y)\}$, i.e. "$z$ is the number of primes of the form $2^n + 1$ that are smaller than x". This counting relation is rudimentary, because $z \leq \log_2(x)$ and $\{y \mid y = 2^n + 1 \text{ and } \textsc{Prime}(y)\}$ is rudimentary. (Indeed, a result proved by Paris

and Wilkie (see [25]) states that $\mathcal{R}$ is closed under polylog counting.) Hence, there is an algorithm in LTH which accepts it, although we cannot immediately produce such an algorithm. Since the dyadic representation of integers $y \geq 5$ of the form $2^n + 1$ is $1 \ldots 121$, there is a linear time non-erasing oracle algorithm computing z. The successive queries are $2, 11, 21, 121, 1121, 11121, \ldots$ (at each step, only the last two digits 21 are modified into 121) and, on a particular tape, a value, starting from 0, is incremented by one after every positive answer to a query. After all queries have been asked, that value is compared with z. Note that, here, the sum of the length of the queries is *quadratic* in the length of x, whereas the number of modifications to the oracle tape is linear, and thus the process runs in linear time.

Another advantage of this new characterization of LTH is to permit local changes on query words providing also the possibility of asking many long queries. On the other hand, any algorithm based on calling the oracle on every integer smaller than the entry remains forbidden.

Definition 3.1 Let us denote by $LTH_{non-eras}$ the class of languages obtained in the same way as LTH, except that one does *not* assume that before the next query is asked, the previous one is erased.

In this section it is established that this model and the original one are equally powerful.

Proposition 3.2 $LTH_{non-eras} = LTH$

Proof: We present below the basic step of the proof : how to replace a non-erasing oracle linear algorithm by a "deeper" erasing oracle linear algorithm. The proof is then completed by induction. Let $\mathcal{M}$ be a linear time non-deterministic non-erasing oracle Turing machine and let $\mathcal{A}$ be its oracle set. We provide a linear time non-deterministic (erasing) oracle machine $\mathcal{M}'$ that accepts the same language as $\mathcal{M}$. For convenience, we will suppose that $\mathcal{M}'$ has one additional tape (compared to $\mathcal{M}$), called tape 0. The main idea is to guess on tape 0 a linear-length word W containing the information about the oracle calls in $\mathcal{M}$:

- what is written at each step on the oracle tape (tape 2),
- what moves are performed by head 2,
- when the oracle is asked a query,
- what are its answers.

Then, the set of words accepted by another non-deterministic linear time erasing machine $\mathcal{A}'$ is used (once) as an oracle to verify that the answers we guessed correspond to the queries we guessed.

Let w be the input of $\mathcal{M}$. The word W that is guessed is of the form : $W = c_0\alpha_0\sharp \ldots \sharp c_m\alpha_m$ with $c_i \in (M \times \{1,2\})^*$ and $\alpha_i \in \{yes, no\}$. Let us denote by w_i the word obtained by performing the actions $c_0, \ldots, c_i$ on an empty tape. Each w_i corresponds to the ith query of $\mathcal{M}$, and α_i corresponds to the oracle's answer. Since $\mathcal{M}$ works in linear time, the length of W is linear in the length of w.
Let us first describe $\mathcal{A}'$:

Input : W (as above)
Find m from W
Guess $i \leq m$ and construct w_i from W.
Query oracle $\mathcal{A}$ about w_i
Accept iff α_i is not the answer of $\mathcal{A}$ on w_i

In other words, $\mathcal{A}'$ refuses W if and only if for every $i \leq m$, the answer given by $\mathcal{A}$ on input w_i is α_i. Because one single query is asked, the linear time algorithm $\mathcal{A}'$ is an erasing one. Again, recall that the sum of the length of the w_i can be more than linear in the length of w (thus also in that of W). Hence $\mathcal{A}'$ is not allowed to verify in turn, for $i = 0$ to m, that α_i is the answer of $\mathcal{A}$ on input w_i. Instead, the strategy of $\mathcal{A}'$ consists in making each test on a separate branch of its computation tree. From now on, let us consider the set of words accepted by $\mathcal{A}'$ as an oracle set, still denoted by $\mathcal{A}'$.

We are interested in the case when $w_0, \ldots, w_m$ are precisely the successive words written on the oracle tape before each oracle call during a computation of $\mathcal{M}$ on input w relative to $\mathcal{A}$. This property has to be checked by $\mathcal{M}'$. We can now give the algorithm for $\mathcal{M}'$.

Input : $w \in \{1,2\}^*$
Guess on tape 0 a word $W = c_0\alpha_0\sharp \ldots \sharp c_m\alpha_m$ as above
Go to the leftmost cell on tape 0

Query the oracle $\mathcal{A}'$ about W
If the oracle's answer is ''yes'', then reject w
Else
- Work as $\mathcal{M}$ and always go to the right on tape 0.
- At each step of $\mathcal{M}$, verify that the move performed by $\mathcal{M}$ on tape 2 is the same as the move read on tape 0 and that the letter written by $\mathcal{M}$ on tape 2 is the same as the letter read on tape 0.
- If $\mathcal{M}$ reaches q_{call}, then verify that some sequence $\alpha_i\sharp$ is written on tape 0 and act as $\mathcal{M}$ according to the answer α_i (move the head of tape 0 to the right of the corresponding $\sharp$ symbol).

Accept w iff $\mathcal{M}$ accepts

$\mathcal{M}'$ is a nondeterministic linear time erasing oracle algorithm (again one single query is asked). The "else" part of $\mathcal{M}'$ deterministically controls that the successive w_i are precisely the successive words written on the oracle tape before each oracle call during a computation of $\mathcal{M}$ on input w relative to $\mathcal{A}$ and simulates $\mathcal{M}$ on w learning from $\mathcal{A}'$ the answers given by the oracle $\mathcal{A}$. We complete the proof by noting that the languages accepted by $\mathcal{M}$ and $\mathcal{M}'$ are the same. □

Altogether, we replace $\mathcal{M}$ and its oracle set $\mathcal{A}$ by $\mathcal{M}'$ and its oracle set $\mathcal{A}'$ which in turn uses $\mathcal{A}$ as oracle. Hence for each non-erasing oracle level we substitute two nested erasing oracle levels. Then, by induction, we have $\Sigma_i^{lin} \subseteq \Sigma_i^{non-eras} \subseteq \Sigma_{2i}^{lin}$. It seems unlikely that $\Sigma_i^{lin} = \Sigma_i^{non-eras}$, but the inclusion $\Sigma_i^{non-eras} \subseteq \Sigma_{i+k}^{lin}$ might hold for some fixed k.

4 The Counting Linear Time Hierarchy

In this section, an extended version of the Linear Time Hierarchy is proposed in order to take counting into account.

Definition 4.1 Let $R(x_1, \ldots, x_p)$ be a relation over integers. A relation $\sharp R(y, x_1, \ldots, x_p)$ is said to be obtained from R by a *counting operation* when for some $i \in \{1, \ldots, p\}$: $\sharp R(y, x_1, \ldots, x_p)$ holds iff y is the number of $x < x_i$ such that $R(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_p)$ holds. In other words, $\sharp R(y, x_1, \ldots, x_p)$ iff $y = \sharp\{x < x_i \ : \ R(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_p)\}$.

As the variables can be permuted, we always count relatively to the first variable x_1.

Definition 4.2 Let $\mathcal{R}^\sharp$ denote the smallest class of relations over integers containing the graphs of addition and multiplication and closed under Boolean operations, explicit transformations, variable bounded quantifications *and* counting operations.

In what follows, we use the fact that $\mathcal{R}^\sharp$ is closed under polynomial substitution (i.e. a polynomial can be substituted for any variable, see [9]). $\mathcal{R}^\sharp$ is more than a simple closure of $\mathcal{R}$ under counting : for example alternation between counting and quantifications is allowed. We thus obtain a wide variety of predicates, such as the set of prime numbers of odd index $(3, 7, 13, \ldots)$.

The model of computation (called $\sharp LTH$) we propose for capturing $\mathcal{R}^\sharp$ naturally extends LTH (as $\mathcal{R}^\sharp$ naturally extends $\mathcal{R}$). The idea is to use oracles that count accepting computations (for a definition of counting classes, such as $\sharp P$, see [31]).

Let us denote by $\sharp LinTime$ the following class of functions : $f \in \sharp LinTime$ iff there exists a nondeterministic linear time Turing machine $\mathcal{M}$ such that on all input $x_1, \ldots, x_p$, $f(x_1, \ldots, x_p)$ is the number of accepting computations of $\mathcal{M}$ on $x_1, \ldots, x_p$. Then, the defintion of $\sharp LTH$ is inductive and follows that of LTH. We set :

$$\Sigma_0^{\sharp lin} = LinTime, \qquad \Sigma_{i+1}^{\sharp lin} = \sharp LinTime(\Sigma_i^{\sharp lin}),$$

$$\sharp LTH = \bigcup_{i \geq 0} \Sigma_i^{\sharp lin}.$$

A function $f \in \Sigma_{i+1}^{\sharp lin}$ is defined via a non-deterministic oracle linear time Turing machine $\mathcal{M}_f$ with *two* oracle tapes (namely tapes 2 and 3). Their query mechanism is as follows : a sequence of integers $x_1, \ldots, x_p$ is written on tape 2 and a single integer y on tape 3. The oracle answer is "yes" iff $y = g(x_1, \ldots, x_p)$ where the oracle function $g \in \Sigma_i^{\sharp lin}$.

We now prove the main result of this section.

Theorem 4.3 $\sharp LTH = \mathcal{R}^{\sharp}$

Note that $\sharp LTH$ is a class of functions whereas $\mathcal{R}^{\sharp}$ is a class of relations. Thus we show on the one hand that the characteristic functions of the relations in $\mathcal{R}^{\sharp}$ are in $\sharp LTH$, and on the other hand that the graphs of the functions in $\sharp LTH$ are in $\mathcal{R}^{\sharp}$. Remember that the characteristic function f_R of a relation $R(x_1, \ldots, x_p)$ is defined as follows : for all input $(x_1, \ldots, x_p)$, $f_R(x_1, \ldots, x_p) = 0$ if$(x_1, \ldots, x_p) \notin R$, and $f_R(x_1, \ldots, x_p) = 1$ otherwise.

Now, let us turn to the proof of Theorem 4.3. The right-to-left inclusion is divided into the two following lemmas.

Lemma 4.4 $\mathcal{R} \subseteq \sharp LTH$

Proof: We prove inductively that if $R(x_1, \ldots, x_p) \in \Sigma_i^{lin}$, then $f_R \in \Sigma_{2i}^{\sharp lin}$.

The basic case $i = 0$ comes from the very definition of $\Sigma_0^{\sharp lin}$.

Now, let $R(x_1, \ldots, x_p) \in \Sigma_{i+1}^{lin} = NLinTime(\Sigma_i^{lin})$, and let $\mathcal{M}$ accepting $R(x_1, \ldots, x_p)$, with an oracle relation $S(y_1, \ldots, y_q) \in \Sigma_i^{lin}$. The induction hypothesis says that $f_S \in \Sigma_{2i}^{\sharp lin}$. Let us modify the algorithm $\mathcal{M}$ as follows : replace each instruction "query $S(y_1, \ldots, y_q)$" by "query $f_S(y_1, \ldots, y_q) = 1$". The obtained algorithm (say $\mathcal{M}'$) is a nondeterministic linear time Turing machine with an oracle in $\Sigma_{2i}^{\sharp lin}$, hence it defines a function $g \in \Sigma_{2i+1}^{\sharp lin}$. Moreover, $\mathcal{M}'$ accepts R as well. Now, let us consider the following algorithm :

Input : $x_1, \ldots, x_p$
  Guess $y > 0$
  Query the oracle g if $y = g(x_1, \ldots, x_p)$
  If the oracle's answer is ''yes'', then accept $x_1, \ldots, x_p$
  Else reject $x_1, \ldots, x_p$

This algorithm has at most one accepting computation and it accepts $x_1, \ldots, x_p$ iff $x_1, \ldots, x_p$ is accepted by $\mathcal{M}'$ i.e. $R(x_1, \ldots, x_p)$ holds. Hence, it precisely defines f_L. Moreover this algorithm is in $\Sigma_{2(i+1)}^{\sharp lin}$ since $g \in \Sigma_{2i+1}^{\sharp lin}$. □

Lemma 4.5 $\sharp LTH$ *is closed under boolean operations, explicit transformation, variable bounded quantification* and *counting.*

Proof: Only the closure under counting is proved here. Closure under variable bounded quantification can be deduced from closure under counting because predicates of the form $\forall x_1 < x_2\ R(x_1, \ldots, x_p)$ are nothing but $x_2 = \sharp\{x_1 < x_2 \mid R(x_1, \ldots, x_p)\}$. The other closure properties are routine matters.

Let $R(x_1, \ldots, x_p)$ be a relation such that $f_R \in \sharp LTH$, i.e. $f_R \in \Sigma_i^{\sharp lin}$ for some i. We have to show that the characteristic function $f_{\sharp R}$ of the relation $\sharp R(y, x_1, \ldots, x_p)$ defined by $\sharp R(y, x_1, \ldots, x_p)$ iff $y = \sharp\{x < x_1 / R(x, \ldots, x_p)\}$ is also in $\sharp LTH$. We prove that $f_{\sharp R} \in \Sigma_{i+2}^{\sharp lin}$.

Let $\mathcal{M}$ be a machine in $\Sigma_i^{\sharp lin}$ which corresponds to f_R. We construct the two following machines $\mathcal{N}$ and $\mathcal{M}'$:

$\mathcal{N} \equiv$
Input : $x_1, \ldots, x_p$
  Guess an integer $x < x_1$.
  Query whether $f_R(x, x_2, \ldots, x_p) = 1$.
  Accept iff oracle answers ''yes''.

$\mathcal{M}' \equiv$
Input : $y, x_1, \ldots, x_p$

Write $x_1, \ldots, x_p$ and y on oracle tapes and ask whether y is the number of accepting computations of $\mathcal{N}$ on $x_1, \ldots, x_p$
Accept iff oracle answers ''yes''

Let us examine $\mathcal{N}$ and $\mathcal{M}'$. First, note that the number of accepting computations of $\mathcal{N}$ is precisely the number of $x < x_1$ for which $R(x, x_2, \ldots, x_p)$ holds. Now, $\mathcal{M}'$ is deteministic and accepts $(y, x_1, \ldots, x_p)$ iff $\sharp R(y, x_1, \ldots, x_p)$ holds, so that it defines $f_{\sharp R}$. □

By definition $\mathcal{R}^\sharp$ is the smallest class of relations closed under boolean operations, explicit transformations, variable bounded quantification and counting. So, $\mathcal{R}^\sharp \subseteq \sharp LTH$.

All that remains is to prove the converse inclusion.

Lemma 4.6 $NLinTime \subseteq \mathcal{R} \subseteq \mathcal{R}^\sharp$

The proof given here mainly follows that of [15]. It is given in detail in view of Lemma 4.7.

Proof: Let $\mathcal{L}$ be a language over the alphabet $\{1, 2\}$ accepted by the non-deterministic linear time machine $\mathcal{M}$, as defined in Section 2. Especially, let k be the number of tapes of $\mathcal{M}$ and l be the total number of symbols of $\mathcal{M}$, i.e. the cardinality of the set $Q \cup M \cup A$. Let $w \in \{1, 2\}^*$ be some word, and let x be the number with dyadic representation w (we denote by $|x|_2$ the length of w). It is shown here that $w \in \mathcal{L}$ iff x satisfies some rudimentary relation of the form $\exists y < x^{\alpha_\mathcal{M}}(\textsc{Code}_\mathcal{M}(y) \wedge \textsc{Init}_\mathcal{M}(x, y) \wedge \textsc{Propag}_\mathcal{M}(x, y) \wedge \textsc{Accept}_\mathcal{M}(x, y))$.

In the formula above, the integer y codes an accepting computation of the machine $\mathcal{M}$ on the input w as follows : the l-ary expansion of y codes a sequence of $(3k+1)$-tuples (one for each computation step). Since $\mathcal{M}$ works in linear time, the length of y is linear in that of x. The $(3k+1)$-tuple of rank t is of the form : $(q^t, (a_1^t, b_1^t, m_1^t), (a_2^t, b_2^t, m_2^t), \ldots, (a_k^t, b_k^t, m_k^t))$ where q^t is the state at time t, a_i^t the symbol read by the ith head, b_i^t the symbol written by the ith head, and m_i^t the move of the head after writing. Hence, if $\beta_\mathcal{M} \times |w|$ is the computation time of the machine $\mathcal{M}$ on the input w, we have $\textsc{Length}(l, y) = \beta_\mathcal{M} \times (3k+1) \times |x|_2$.

Since we cannot encode the space-time diagram into a linear-length integer, the code y of the computation is only partial, and we use arithmetic to complete the missing part. More precisely, we need to compute the content of the cell visited by a given head at a given time.

We can access the information concerning the computation of $\mathcal{M}$ encoded in y via the rudimentary predicate $\textsc{Digit}(l, y, d)$, because for all $0 \leq t < \beta_\mathcal{M}|x|_2$, and for all $1 \leq h \leq k$ we have :

$$\begin{array}{ll} \textsc{Digit}(l, y, (3k+1)t + 3h) = m_h^t, & \textsc{Digit}(l, y, (3k+1)t + 3h - 2) = a_h^t, \\ \textsc{Digit}(l, y, (3k+1)t + 3h - 1) = b_h^t, & \textsc{Digit}(l, y, (3k+1)t) = q^t. \end{array}$$

In other words, in the l-ary representation of y, the $3k+1$ digits of rank 0 to $3k$ correspond to time 0, that of rank $3k+1$ to $6k+1$ to time 1, ..., that of rank $(3k+1)t$ to $(3k+1)t + 3k$ to time t, ...

For a given tape h $(1 \leq h \leq k)$, in order to compute the position of the head at time t, we have to determine the difference between the number of `r` and the number of `l` in the sequence $m_h^0, \ldots, m_h^t$. This is done by the following function, defined by induction, for all $1 \leq h \leq k$:

$\textsc{Pos}(y, h, 0) = 0$,

$$\textsc{Pos}(y, h, t+1) = \begin{cases} \textsc{Pos}(y, h, t) & \text{if} \quad m_h^t = \mathtt{m} \\ \textsc{Pos}(y, h, t) + 1 & \text{if} \quad m_h^t = \mathtt{r} \\ \textsc{Pos}(y, h, t) - 1 & \text{if} \quad m_h^t = \mathtt{l} \quad \text{and} \quad \textsc{Pos}(y, h, t) > 0 \\ \textsc{Pos}(y, h, t) & \text{if} \quad m_h^t = \mathtt{l} \quad \text{and} \quad \textsc{Pos}(y, h, t) = 0. \end{cases}$$

The graph of this function is in *LogSpace* so it is rudimentary. Moreover, the function Pos computes as expected the position of the head h at time t.

Now we can define the relation $\textsc{LastTime}(y, h, t, t')$ ($t' < t$ is the last time when the head h was in the same position as at time t) as follows :

$$t' < t \wedge \textsc{Pos}(y, h, t) = \textsc{Pos}(y, h, t') \wedge \forall t'' < t(t' < t'' \rightarrow \neg \textsc{Pos}(y, h, t'') = \textsc{Pos}(y, h, t)).$$

The letter read at time t on tape $h > 1$ is the letter written on tape h during the previous visit of the same cell. If the cell visited at time t had never been visited before then one reads B. Concerning the read-only input tape, the letter read at time t is always the symbol that was originally written in

the cell of index $\textsc{Pos}(y,1,t)$. Hence we can define the fundamental relation $\textsc{ReadLet}(x,y,h,t,a)$: the letter read at time t on tape h is a.

For $h=1$, $\textsc{ReadLet}(x,y,1,t,a)$ iff

$$\forall p < y(p = \textsc{Pos}(y,1,t) \rightarrow ((p \leq |x|_2) \wedge a = \textsc{Digit}(2,x,p)) \vee (p > |x|_2 \wedge a = B))),$$

and for $h>1$, $\textsc{ReadLet}(x,y,h,t,a)$ iff

$$(\forall t' < t \neg \textsc{LastTime}(y,h,t,t') \wedge a = B) \quad \vee \exists t' < t(\textsc{LastTime}(y,h,t,t') \wedge a = b_h^{t'}).$$

We are now ready to complete the proof. We define as expected the relation $\textsc{Code}_{\mathcal{M}}(y)$ stating that y codes a computation of the machine $\mathcal{M}$ and the three relations encoding the computation : $\textsc{Init}_{\mathcal{M}}(x,y)$, $\textsc{Accept}_{\mathcal{M}}(x,y)$ and $\textsc{Propag}_{\mathcal{M}}(x,y)$. We have :

$\textsc{Code}_{\mathcal{M}}(y) \equiv \forall t < \beta_{\mathcal{M}}|x|_2 \;\; (q^{t+1}, b_1^t, \ldots, b_k^t, m_1^t, \ldots, m_k^t) \in \delta(q^t, a_1^t, \ldots, a_k^t)$ (it is a finite list of possible values),

$\textsc{Init}_{\mathcal{M}}(x,y) \equiv q^0 = q_0 \quad \wedge a_1^0 = \textsc{Digit}(2,x,0) \wedge \bigwedge_{h=2}^{k} a_h^0 = B$ (initial configuration),

$\textsc{Accept}_{\mathcal{M}}(x,y) \equiv \forall t < y \; (t = \beta_{\mathcal{M}}|x|_2 \rightarrow q^t = q_a)$ (acceptance),

$$\textsc{Propag}_{\mathcal{M}}(x,y) \equiv \forall t < \beta_{\mathcal{M}}|x|_2 \bigwedge_{h=1}^{k} a_h^t = \textsc{ReadLet}(x,y,h,t) \quad \text{(propagation)}.$$

Finally, in view of Lemma 4.7, note that the correspondence between the accepting computations of $\mathcal{M}$ on the entry x and the integers $y < x^{\alpha_{\mathcal{M}}}$ satisfying $\textsc{Code}_{\mathcal{M}}(y) \wedge \textsc{Init}_{\mathcal{M}}(x,y) \wedge \textsc{Propag}_{\mathcal{M}}(x,y) \wedge \textsc{Accept}_{\mathcal{M}}(x,y)$ is one-to-one and onto. □

Lemma 4.7 $\sharp LTH \subseteq \mathcal{R}^{\sharp}$

Proof: By induction. Let $f \in \Sigma_i^{\sharp lin}$ be defined by some machine $\mathcal{M}$. It is shown, for every i, that there exists a predicate $R(x,y) \in \mathcal{R}^{\sharp}$ s.t. the number z of accepting computations of $\mathcal{M}$ on input w_x is equal to the number of $y < x^{\beta_{\mathcal{M}}}$ satisfying $R(x,y)$. In other words, $z = f(x)$ iff $z = \sharp\{y < x^{\beta_{\mathcal{M}}} : \; R(x,y)\}$.

For $i=0$, f is the characteristic function of a language in $LinTime$. Let us go back to the proof of Lemma 4.6. Since $\mathcal{M}$ is deterministic, the condition above is fulfilled.

Now, assume the property is true for i. Let $f \in \Sigma_{i+1}^{\sharp lin}$ corresponding to some non-deterministic linear time machine $\mathcal{M}$ with an oracle function $g \in \Sigma_i^{\sharp lin}$. By the induction hypothesis, the binary relation $y = g(x)$ is in $\mathcal{R}^{\sharp}$. Let us go back again to the proof of Lemma 4.6.

A relation $\textsc{Written}(x,y,t,h,u)$ telling us that the dyadic representation of u is written on tape h at time t has to be defined (it will be used only for $h=2,3$). The maximal length p of the word written on tape h at time t is obtained using the relation $\textsc{MaxPos}(x,y,h,t,p)$ defining the maximal position reached on tape h by the head before time t. Hence we have $\textsc{MaxPos}(x,y,h,t,p)$ iff

$$\exists t' < t \; \textsc{Pos}(x,y,h,t',p) \quad \wedge \forall t'' < t \forall q < y \; [\textsc{Pos}(x,y,h,t'',q) \rightarrow q \leq p].$$

And for all $0 \leq q \leq p$, we need the relation $\textsc{LastPos}(x,y,h,q,t,t')$ defining the last time $t' < t$ when the head of tape h was in position q. Hence we have $\textsc{LastPos}(x,y,h,q,t,t')$ iff

$$t' < t \wedge \textsc{Pos}(x,y,h,t',q) \quad \wedge \; \forall t'' < t \; (t'' > t' \rightarrow \neg \textsc{Pos}(x,y,h,t'',q)).$$

Finally, the relation $\textsc{Written}(x,y,h,t,u)$ can be defined (if $h \neq 1$) by the following formula :

$$\exists p < y \;\; [\textsc{MaxPos}(x,y,h,t,p) \wedge |u|_2 \leq p$$

$$\wedge \forall q \leq p \; \exists t' < t \;\; (\textsc{LastPos}(x,y,h,q,t,t') \wedge \textsc{Digit}(2,u,q) = b_h^{t'})].$$

Now, let us deal with the oracle calls as follows :

- The relations $\textsc{Init}_{\mathcal{M}}(x,y)$, $\textsc{Accept}_{\mathcal{M}}(x,y)$ and $\textsc{Propag}_{\mathcal{M}}(x,y)$ remain unchanged.
- The relation $\textsc{Code}_{\mathcal{M}}(y)$ is replaced by the following relation $\textsc{Newcode}_{\mathcal{M}}(x,y)$:

$$\begin{array}{ll} \forall t < \beta_{\mathcal{M}}|x|_2 & (\neg q^t = q_{call} \rightarrow (q^{t+1}, b_1^t, \ldots, b_k^t, m_1^t, \ldots, m_k^t) \in \delta(q^t, a_1^t, \ldots, a_k^t)) \wedge \\ & \big(q^t = q_{call} \rightarrow \forall u < y \forall v < y \; (\textsc{Written}(x,y,2,t,u) \wedge \textsc{Written}(x,y,3,t,v) \longrightarrow \\ & (v = g(u) \rightarrow q^{t+1} = q_{yes} \wedge \bigwedge_{h=1}^{k} m_h^t = \mathtt{m} \wedge \bigwedge_{h=1}^{k} b_h^t = a_h^t) \wedge \\ & (\neg v = g(u) \rightarrow q^{t+1} = q_{no} \wedge \bigwedge_{h=1}^{k} m_h^t = \mathtt{m} \wedge \bigwedge_{h=1}^{k} b_h^t = a_h^t))). \end{array}$$

□

Note that, considering ordinary oracles instead of counting oracles, this proof can also be seen as an arithmetical proof of Proposition 3.2, since it works equally well for erasing machines and for non-erasing machines.

5 On Majority Operations

It is well known that majority and exact counting often have the same expressive power (for instance $P(PP) = P(\sharp P)$, see [1]). The aim of this section is to compare the expressive powers of these two notions within the framework of linear time algorithms. We first consider the case of bounded arithmetics.

Definition 5.1 Let $R(x_1, \ldots, x_p)$ be a rudimentary relation. A relation $R'(x_1, \ldots, x_p)$ is said to be obtained from R by a *majority operation* when for some $i \in \{1, \ldots, p\}$: $R'(x_1, \ldots, x_p)$ holds iff there are more than $\frac{1}{2}x_i$ integers x such that $x < x_i$ and $R(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_p)$ holds. In what follows, we will denote by $\mathbf{Maj}_{x_i} R(x_1, \ldots, x_p)$ such a predicate $R'(x_1, \ldots, x_p)$.

Note that it is possible to consider this majority operation as a generalized quantifier. The formulas thus obtained are easier to handle and to design than those using exact counting. In particular, a prenex normal form exists, using three types of quantifiers $(\exists, \forall, \mathbf{Maj})$.

Definition 5.2 Let $\mathcal{R}^{Maj}$ denote the smallest class of relations over integers containing the graphs of addition and multiplication and closed under Boolean operations, explicit transformations, variable bounded quantifications *and* majority operations.

Note that, by definition, we have $\mathcal{R} \subseteq \mathcal{R}^{Maj}$. We show that, for rudimentary predicates, knowing if more than half of the possible values of a variable x satisfies a given predicate R is exactly as powerful as knowing the exact number of values of x for which R holds.

Proposition 5.3 $\mathcal{R}^{\sharp} = \mathcal{R}^{Maj}$

Proof: The right to left inclusion is straightforward. For the converse one, we only have to show that $\mathcal{R}^{Maj}$ is closed under counting, i.e. that counting operations can be replaced by majority operations.

Let y and m be two integers with $y < 2^m$ (i.e. $m \geq \textsc{Length}(y, 2)$) and let $y = \alpha_{m-1}2^{m-1} + \alpha_{m-2}2^{m-2} + \cdots + \alpha_0 2^0$ be the binary expansion of y (i.e. $\alpha_i = \textsc{Digit}(2, y, i)$ for all $i < m$).

For every $y, m \geq \textsc{Length}(y, 2)$ and $i < m$, the following function f (whose graph is rudimentary) is defined : $f(y, m, i) = (1 - \alpha_{m-1})2^{m-1} + \cdots + (1 - \alpha_{i+1})2^{i+1} + 2^i$. Thus, denoting $1 - \alpha$ by $\overline{\alpha}$, we have :
$f(y, m, 0) = \overline{\alpha}_{m-1}2^{m-1} + \cdots + \overline{\alpha}_1 2^1 + 2^0$,
$f(y, m, 1) = \overline{\alpha}_{m-1}2^{m-1} + \cdots + \overline{\alpha}_2 2^2 + 2^1$,
$\ldots$,
$f(y, m, m-2) = \overline{\alpha}_{m-1}2^{m-1} + 2^{m-2}$,
and $f(y, m, m-1) = 2^{m-1}$.
Thus $1 \leq f(y, m, i) < 2^m$, for all $i < m$. We next use the function f to determine y bit by bit. Indeed, for all $i < m$, we have $\alpha_i = 1$
iff $y - (\alpha_{m-1}2^{m-1} + \cdots + \alpha_{i+1}2^{i+1}) \geq 2^i$
iff $y \geq \alpha_{m-1}2^{m-1} + \cdots + \alpha_{i+1}2^{i+1} + 2^i$
iff $y + f(y, m, i) \geq 2^{m-1} + \cdots + 2^{i+1} + 2^i + 2^i = 2^m$.

Finally, note that the majority predicate $\mathbf{Maj}_{x_1} R(x_1, x_2, \ldots, x_p)$ with $x_1 = 2^{m+1} - 1$ is true iff the number of integers $x < x_1$ satisfying $R(x, x_2, \ldots, x_p)$ is $\geq 2^m$.

Using the two previous remarks, we replace the counting operation $y = \sharp\{x < x_1 \mid R(x, x_2, \ldots, x_p)\}$ by the following majority formula :

$$\begin{array}{l}\exists m \leq x_1 \exists z \leq 4x_1 \ \{\ m = \textsc{Length}(x_1, 2) \wedge z = 2^{m+1} - 1 \wedge \\ \forall i \leq m - 1 \\ \mathbf{Maj}_z[(z < x_1 \wedge R(z, x_2, \ldots, x_p)) \vee (2^m \leq z < 2^m + f(y, m, i))] \leftrightarrow \textsc{Digit}(2, y, i) = 1\ \}.\end{array}$$

Note that a linearly bounded quantification is used (namely $\exists z \leq 4x_1$, this is because $z = 2^{m+1} - 1$ and $2^{m-1} \leq x_1$). This is not allowed by the very definition of $\mathcal{R}^{Maj}$. To overcome this problem it suffices to first carefully separate each counting operation into four smaller ones. E.g. if $\frac{x_1}{4}$ is an integer, write $y = y_0 + y_1 + y_2 + y_3$ with $y_i = \sharp\{j < \frac{x_1}{4} \mid R(j + i\frac{x_1}{4}, x_2, \ldots, x_p)\} \leq \frac{x_1}{4}$ for $i = 0 \ldots 3$, and substitute $\frac{x_1}{4}$ for x_1 in the four corresponding majority formulas. The other cases are similarly dealt with. □

For Turing machines, probabilistic acceptance can model majority. Hence, the link between counting and majority in bounded arithmetics naturally induces a characterization of $\sharp LTH$ in terms of probabilistic oracle computations.

Remember that PP (Probabilistic P) is the class of language recognized by polynomial time nondeterministic machines that accept x if and only if more than half of the computation paths accept x. Computation trees of such machines are required to be complete binary trees. Let us denote by $PLinTime$ the class obtained by restricting PP to languages recognized in linear time and by $PLTH$ the class obtained similarly to $\sharp LTH$ by taking the union of the following classes Σ_i^{Plin} :

$$\Sigma_o^{Plin} = LinTime,\ \Sigma_{i+1}^{Plin} = PLinTime(\Sigma_i^{Plin}).$$

The class $PLTH$ is the linear equivalent to Toran's counting class (see [30]).

Theorem 5.4 $\mathcal{R}^{Maj} = \mathcal{R}^{\sharp} = \sharp LTH = PLTH$

Proof: It remains to prove the last equality. The inclusions $PLTH \subseteq \mathcal{R}^{\sharp}$ and $\mathcal{R}^{Maj} \subseteq PLTH$ follow from minor modifications of the proof of Theorem 4.3. □

6 Conclusion

Let us list some open problems about LTH and $\sharp LTH$. Of course, the basic question is whether LTH is equal to $\sharp LTH$, and this is a question that might be very difficult to solve. Since quantifications are particular cases of counting, it would be interesting to know how many alternations of quantifications can be replaced by one single counting. In the same context, the question could be raised as to whether an analogue to Toda's theorem ($PH \subseteq P(\sharp P)$, see [29]) holds for the Linear Time Hierarchy i.e. if $LTH \subseteq LinTime(\sharp LinTime)$.

Also, we know that $NLIN \subseteq LTH$ and more precisely that $NLIN \subseteq \Sigma_2^{lin}$. Hence a natural question holds : is there a type of rudimentary formula corresponding to $NLIN$?

Another question is the existence of a normal form for algorithms in $\sharp LTH$ similar to the logical normal form for algorithms in LTH (see for instance [15]).

Now, for variant machine models, it would be interesting to provide a characterization of $\sharp LTH$ in terms of alternating Turing machines or stack register machines, thus making the relationship with the results presented in [7] more precise. Still within the framework of alternative definitions, it would be interesting to find a logical characterization of $\sharp LTH$, or more likely of $PLTH$.

References

[1] D. Angluin, On counting problems and the polynomial-time hierarchy, *Theoretical Computer Science*, 12, 1980, pp. 161-173.

[2] A. Bel'tyukov, A computer description and a hierarchy of initial Grzegorczyk classes, *J. Sov. Math.*, 20, 1982, pp. 2280-2289. Translation from *Zap. Nauk. Sem. Lening. Otd. Mat. Inst., V.A. Steklova AN SSSR*, 88, 1979, pp. 30-46.

[3] A. Bel'tyukov, One-way one-dimensional cellular automata and small subrecursive classes, *Communication JAF'15*, Mons, Belgium, 1997, pp. 21-27.

[4] J.H. Bennett, *On Spectra*, Doctoral Dissertation, Princeton University, Princeton, N.J., 1962.

[5] S. A. Bloch, J. F. Buss, J. Goldsmith, Sharply Bounded Alternation and Quasilinear Time, *Theory of Computing Systems*, 31(2), 1998, pp.187-214.

[6] J. F. Buss, Relativized Alternation and Space-Bounded Computation, *J. of Comput. and Syst. Sci.* 36, 1998, pp. 351-378.

[7] P. Clote, Nondeterministic stack register machines, *Theoretical Computer Science A*, 178, 1997, pp. 37-76.

[8] A. Durand, C. Lautemann, M. More, Counting results in weak formalisms, *preprint SDAD*, 98-14, 1998.

[9] H.A. Esbelin, M. More, Rudimentary relations and primitive recursion : a toolbox, *Theoretical Computer Science*, 193, 1998, pp. 129-148.

[10] E. Grädel, On the notion of linear time computability, *International J. of Foundations of Computer Science*, 1, 1990, pp. 295-307.

[11] E. Grandjean, Linear-time algorithms and NP-complete problems, *SIAM J. Comput.*, 23, 1994, pp. 573-597.

[12] E. Grandjean, Sorting, linear-time and the satisfiability problem, *Annals of Math. and Artificial Intelligence*, 16, 1996, pp. 183-236.

[13] E. Grandjean, F. Olive, Monadic logical definability of nondeterministic linear time, *Comput. Complex.* , 7, 1998, pp. 54-97.

[14] Y. Gurevich, S. Shelah, Nearly linear time, *Lect. Notes Comput. Sci.* , 363, 1989, Springer-Verlag, pp. 108-118.

[15] P. Hájek, P. Pudlák *Metamathematics of First-Order Arithmetic*, Springer-Verlag, Berlin, 1993.

[16] K. Harrow, *Sub-elementary Classes of Functions and Relations*, Doctoral Dissertation, New-York University, Department of Mathematics, 1973.

[17] K. Harrow, Small Grzegorczyk classes and limited minimum, *Z. Math. Logik Grundlagen Math.*, 11, 1975, pp. 417-426.

[18] N.D. Jones, Context-free languages and rudimentary attributes, *Math. System Theory*, 3, 1969, pp. 102-109.

[19] M. More, F. Olive, Rudimentary Languages and Second-Order Logic, *Mathematical Logic Quarterly*, 43, 1997, pp. 419-426.

[20] G. Myhill, Linear bounded automata, *Wright Air Devel. Div.*, Tech. Note 60-165, Wright-Patterson Air Force Base, OH, 1960.

[21] A.V. Naik, K.W. Regan, D. Sivakumar, On quasilinear-time complexity theory, STACS '94 (Caen, 1994), *Theoret. Comput. Sci.*, 148(2), 1995, pp.325-349.

[22] V.A. Nepomnjascii, Rudimentary predicates and Turing calculations, *Soviet Math. Dokl.*, 11(6), 1970, pp. 1462-1465.

[23] J.B. Paris, W.G. Handley, A.J. Wilkie, Characterizing some low arithmetic classes, *Colloquia mathematica societatis Janos Bolyai*, 1987, pp. 353-365.

[24] J.B. Paris, A.J. Wilkie, Counting problems in bounded arithmetics, *Methods in mathematical logic* (proceedings, Caracas, 1983), *Lecture Notes in Mathematics*, 1130, Springer-Verlag, Berlin, 1985, pp. 317-340.

[25] J.B. Paris, A.J. Wilkie, Counting Δ_0 sets, *Fundamenta Mathematicae*, 127, 1987, pp. 67-76.

[26] C.P. Schnorr, Satisfiability is quasi-linear complete in NQL, *J. ACM*, 25, 1978, pp. 136-145.

[27] R. Smullyan, *Theory of Formal Systems*, Annals of Math. Studies, 47, Princeton University Press, Princeton, N.J., 1961.

[28] L.J. Stockmeyer, The polynomial time hierarchy, *Theor. Comput. Sci.*, 3, 1976, pp. 1-22.

[29] S. Toda, On the computational power of PP and $\oplus$P, *IEEE Symposium on Foundations of Computer Science*, 1989, pp. 514-519.

[30] J. Torán, Complexity classes defined by counting quantifiers, *J. ACM*, 38(3), 1991, pp. 753-774.

[31] L. Valiant, The complexity of enumeration and reliability problems, *SIAM Journal of Computing*, 8(3), 1979.

[32] K. W. Wagner, The complexity of combinatorial problems with succint input representation, *Acta Informatica*, 23, 1986, pp. 325-356.

[33] C. Wrathall, Rudimentary predicates and relative computation, *SIAM J. Comput.*, 7(2), 1978, pp. 194-209.