Counting Results in Weak Formalisms*

Arnaud Durand[1], Clemens Lautemann[2], and Malika More**[3]

[1] Équipe de Logique Mathématique UMR 7056, Université Denis Diderot - Paris 7, 2 place Jussieu, F-75251 Paris Cedex 05, France, durand@logique.jussieu.fr
[2] Institut für Informatik, Johannes Gutenberg-Universität, Mainz, Germany.
[3] Univ Clermont 1, Laboratoire LAIC, IUT Informatique, Campus des Cézeaux, B.P. 86, F-63172, Aubière Cedex, France, more@laic.u-clermont1.fr.

Abstract. The counting ability of weak formalisms is of interest as a measure of their expressive power. The question was investigated in several papers in complexity theory [ABO84,FKPS85,DGS86] and in weak arithmetic [PW87]. In each case, the considered formalism (AC^0–circuits, first–order logic, Δ_0, respectively) was shown to be able to count precisely up to a polylogarithmic number. An essential part of each of the proofs is the construction of a 1–1 mapping from a small subset of $\{0, \ldots, N-1\}$ into a small initial segment. In each case the expressibility of such a mapping depends on some strong argument (group theoretic device or prime number theorem) or intricate construction. We present a coding device based on a collision-free hashing technique, leading to a completely elementary proof for the polylog counting capability of first–order logic (with built–in arithmetic), AC^0–circuits, rudimentary arithmetic, the Linear Hierarchy, and monadic–second order logic with addition.

Note. This paper was written in 1998 and has never been published in a journal nor presented to a conference. Up until now, it was hardly available as a research report. We believe these proceedings are the right place to publish it eventually, because most participants of the Dagstuhl seminar "Circuits, Logic and Games" were friends, colleagues or students of Clemens, who passed away in April 2005. Undoubtedly, Clemens would have been one of the prominent participants of this meeting at which the feeling of his absence was widely shared.

1 Introduction

Proving nontrivial lower bounds on the computational resources needed to solve interesting problems remains the main challenge in complexity theory. In an attempt to gain a deeper understanding of the difficulty of this problem, researchers have investigated a large number of restrictions on computations. Their aim was to find strong enough restrictions to enable them to prove that certain computational problems are not solvable in the resulting weak formalisms. However,

* Collaboration supported by a PROCOPE grant
** Corresponding author.

in most cases where this approach succeeded, lower bounds were proved for problems which are computationally trivial, such as PARITY, the problem of deciding whether the number of 1's in a 0–1–string is odd or even. This and other examples indicate that those formalisms for which we can prove lower bounds are far from being powerful enough to express computation. But how can we measure this distance? One indicator for the computing power of a formalism is its *counting* ability. The more precisely we can count the number of 1's in a string the closer we get to real computing power.

But also without a complexity theoretic motivation, the counting power of formalisms is of interest as a measure of *expressive* power. In fact, within a short period of time, the question was investigated in several papers in complexity theory [ABO84,FKPS85,DGS86] and in weak arithmetic [PW87]. In each case, the result was that the formalism under consideration (AC^0–circuits, first–order logic, Δ_0, respectively) is able to count precisely up to a polylogarithmic number. The proofs, although obtained independently, are structured similarily. One major ingredient in all of them is a mapping which maps all elements of a small subset of $\{0, \ldots, N-1\}$ in a 1–1 fashion into a small initial segment. In each case the expressibility of such a mapping by the given means depends on something strong: Fagin et al. use a group theoretic device, whereas the other papers make use of the prime number theorem. In [PW87] the question is raised whether the proof can be given without employing any such proof–theoretically strong arguments and an alternative but quite intricate proof is proposed.

In this paper, we give a completely elementary proof for the polylog counting capability of constant–depth circuits (AC^0), first–order logic with built–in arithmetic ($FO[BIT]$), rudimentary arithmetic ($\mathcal{R}$), the Linear Hierarchy ($LinH$), and monadic second–order logic with addition ($MSO[+]$)[4]. As explained below, all these formalisms are closely related. Indeed, it would suffice to give the proof for $FO[BIT]$, say, and then employ a general translation to obtain the result for the other systems. However, it is interesting to see that the idea of the proof can be implemented in all these formalisms with the same ease, and a direct proof leads to a construction which is far simpler than the one that would result from applying a general translation. Therefore, after giving the proof in the first–order setting, we will also present the construction as a $LinH$ algorithm. As an application, in Section 3, we will use the counting ability of first–order logic to reprove the fact, earlier shown by Grohe and Hella (personal communication), that BIT–invariant $FO[BIT]$–formulae are not Gaifman local.

2 Presentation of the counting method

In this section, we will first define the different formalisms we are interested in and compare them with respect to expressive power. Finally, we will present a general sketch of our counting method.

Let us denote by $LinTime$ (resp. $NLinTime$) the class of binary languages accepted by a deterministic (resp. non-deterministic) multi-tape Turing machine

[4] For precise definitions see below.

in linear time. For all $k \geq 2$, let AL_k be the set of binary languages accepted in linear time by an alternating Turing machine with $k-1$ alternations, and let $AL_0 = LinTime$ and $AL_1 = NLinTime$.

Definition 1. *The Linear Hierarchy (LinH) is the union of the classes AL_k, for $k \geq 0$.*

An equivalent characterization of $LinH$ (see [Wra78]) is obtained via a linear analogue to the well–known Polynomial Hierarchy (see [Sto76]).

Definition 2. *Let σ be a signature. We call $MSO[\sigma]$ (resp. $FO[\sigma]$) the set of monadic–second order (resp. first-order) formulas of signature σ. I.e. the formulas in $SO[\sigma]$ are of the form $Q_1U_1 \ldots Q_pU_p\Phi$, where $Q_i \in \{\forall, \exists\}$, each U_i is a unary predicate symbol and Φ is a first-order formula of signature $\sigma \cup \{U_1 \ldots, U_p\}$. We say that a set of finite σ–structures is definable in $MSO[\sigma]$ (resp. in $FO[\sigma]$) iff it is the set of finite models of some sentence in $MSO[\sigma]$ (resp. in $FO[\sigma]$).*

In the sequel, we are interested in $MSO[+]$ and $FO[BIT]$ over *words*. Here the structures are of type $\langle\{0, \ldots, |w|-1\}, +, 1_w\rangle$ or $\langle\{0, \ldots, |w|-1\}, BIT, 1_w\rangle$, where $1_w(i)$ is true iff the ith letter of w is a 1, $|w|$ denotes the length of w, and the built-in relations are $+$ (addition) and BIT, respectively. ($BIT(x, y)$ holds iff the bit of rank y of x is 1.) If a set of words is definable in $MSO[+]$ or $FO[BIT]$, we say for short that it is in $MSO[+]$ or $FO[BIT]$.

Definition 3. *A first-order formula in the language of aritheoremetic $\{+, \times\}$ is called Δ_0 if all its quantifiers occur bounded (e.g. $\forall x < y^2$). A* rudimentary *relation over $\mathbb{N}$ is one that can be defined by a Δ_0–formula.*

To any integer x one can associate a word $w \in \{0,1\}^*$ that encodes the binary expansion[5] of x (which we will write starting with the least significant bit) or equivalently a structure $\langle\{0, \ldots, |w|-1\}, +, 1_w\rangle$. It has been proved that, under these correspondences between integers, words and structures, $\mathcal{R}$, $LinH$ and $MSO[+]$ have the same expressive power (see [Wra78], [MO97]).

Now, let us turn to the two other formalisms under consideration, i.e. AC^0 and $FO[BIT]$.

Definition 4. *AC^0 is the class of languages over $\{0,1\}$ accepted by a uniform family of circuits of constant depth and polynomial size.*

It is well known (see [Imm87], for instance) that, given an appropriate notion of uniformity, AC^0 and $FO[BIT]$ over words have exactly the same expressive power, with the same correspondence as above between words and structures. Hence we have $FO[BIT] = AC^0$.

[5] In order to get a one-to-one and onto correspondence between words and integers, it would be preferable to use *dyadic* notation (i.e. words over $\{1,2\}$). Since we will only consider words of the same length, however, binary notation is good enough.

Comparing the expressive powers of the two groups of formalisms, it is easy to see that $FO[BIT] \subseteq MSO[+]$, since, on the one hand, the BIT relation is first-order definable from $+$ and $\times$ and conversely (cf. [Imm99,DDLW98]), and, on the other hand, $\times$ is definable from $+$ using existentially quantified unary variables (see [?]). In the languages framework, it is known that $AC^0 \subseteq LOGSPACE \subseteq LinH$ (see [Nep70,HP93] for the second inclusion), and that AC^0 is more restrictive than $LinH$. For instance, the language $L_{odd} = \{w \in \{0,1\}^* \ / \ w \text{ contains an odd number of 1's }\}$ is not in AC^0 (cf. [FSS81]), whereas it does lie in $LinH$. More generally, counting the number of 1's in words is very easy in $LinH$, but it is not an AC^0 operation.

There is also a translation in the other direction: any unary Δ_0-formula $\Phi(N)$ is easily translated into a $FO[BIT]$-sentence Φ^* by replacing all quantifications $\forall x < N$ by $\forall x$ and $\exists x < N$ by $\exists x$, and other similar syntactic modifications. In such a case, we obtain that $\Phi(N)$ is true iff $\langle \{0, \ldots, N-1\}, BIT, \emptyset \rangle \models \Phi^*$. Here, the free unary second-order variable is interpreted as $\emptyset$, so that the structure can be viewed as an integer written in unary. Thus rudimentary sets exactly correspond to the sets of *integers* (not words) definable in unary by $FO[BIT]$–formulae.

Now, let us turn to counting. We are actually considering two different problems when talking about counting abilities: in the $LinH - MSO[+] - \mathcal{R}$ framework, given an integer N, one counts the number of integers smaller than N that satisfy some previously defined property, whereas in the $AC^0 - FO[BIT]$ framework, given a 0–1–word w, one counts the number of 1's in w.
We are interested in a common positive result: $LinH$, AC^0 et al. are closed under polylog counting. In this paper we present a new and elementary proof of the following Theorem 1 (in Section 3, within the formalism of $FO[BIT]$) and Theorem 2 (in Section 4, using alternating algoritheorems). The method we propose is uniform and directly applies in the different contexts. Notice that, in contrast, a negative result is only known in the AC^0 formalism: AC^0 (resp. $FO[BIT]$) does not express parity (since $L_{odd} \notin AC^0$) and *a fortiori* does not express counting (see [Ajt83,?,?,DGS86]), but both questions remain open for $LinH, \mathcal{R}$ and $MSO[+]$.

Let us denote by $\lg N$ the length of the binary expansion of N.

Theorem 1. *([ABO84,FKPS85,DGS86]) For every integer k, there exists an AC^0 family of circuits such that: the circuit with N inputs has $(\lg N)^k + 1$ outputs and the yth output is 1 iff $y = \sharp\{i < N \ / \ \text{the } i\text{th entry is } 1\}$ and $y \leq (\lg N)^k$.*

Theorem 2. *([PW87]) Let $P(N, \boldsymbol{u})$ be a rudimentary relation. Then, for all fixed k, the counting relation $CP(N, y, \boldsymbol{u}) \equiv \big(\ y = \sharp\{i < N / P(i, \boldsymbol{u})\} \ \wedge \ y \leq (\lg N)^k \ \big)$ is also rudimentary.*

Finally, let us present our coding device. Similarly to the original proofs, the basic idea of our method is to take advantage of the fact that good integers (those that we want to count) are few and encode them into smaller integers, and then to count these codes. The difference is that our proof is entirely based

on the binary representation of integers and uses a new collision-free hashing technique.

Let us introduce some notation. We are interested in the number y of integers x which satisfy some property and are smaller than some bound N (so-called *good* integers). Let $l = \lg N$ and assume that $y \leq l^k$, for some fixed k. We shall use the fact that, under the additional assumption that for some $0 < \varepsilon < 1$ and for any good x, we have $\lg x < l^\varepsilon$, there are standard techniques (namely Lemmas 3 and 5) performing the computation.

1 - *Integers smaller than N are cut into small blocks.*
For instance, any integer $x < N$ is viewed as a list of $l^{\frac{1}{2}}$ blocks of 0-1−strings, each of length $l^{\frac{1}{2}}$, i.e. $x = x_0 * \cdots * x_{l^{\frac{1}{2}}-1}$. We say that z is a *good* block if it is a block in some good integer.

2 - *Good blocks are few and small : they can be numbered.*
There are no more than l^{k+1} good blocks, each of length $l^{\frac{1}{2}}$. Hence, using the standard lemmas, we count good blocks, or equivalently, assign a number to each good block.

3 - *Good integers are encoded by the list of the numbers of their blocks.*
We replace each x_i by its number α_i and define $c(x) = \alpha_0 * \cdots * \alpha_{l^{\frac{1}{2}}-1}$. Hence the code c is a function mapping good integers to binary words of length at most $(k+1)l^{\frac{1}{2}} \lg l$.

This code is one-to-one (i.e. collision-free, this is necessary), but not onto (there are lists of numbers of good blocks that do not correspond to any good integer). Let us call *good* a code that actually encodes a good integer. The number of good integers is clearly equal to the number of good codes.

4 - *Good codes are few and small: they can be counted.*
We already know that the number of good codes is $y \leq l^k$. Moreover, $(k+1)l^{\frac{1}{2}} \lg l$ is eventually smaller than (say) $l^{\frac{3}{4}}$. Thus, the standard lemmas work again.

3 Counting in first–order logic

3.1 Proof in the $FO[BIT]$–setting

We will frequently make use of the fact that $FO[BIT] = FO[BIT, +, \times, <]$.

As explained above, our main tool, which we will use frequently, is the decomposition of the binary representation of a number $y < N$ into blocks of 0-1–strings, all (except, possibly, for the last) of some fixed length L. We number these blocks from 0 to $\lceil \frac{\lg N}{L} \rceil - 1$, beginning with the least significant bits. For the sake of simplicity we will henceforth assume that L divides $\lg N$, the general case can be dealt with by some obvious small modifications.

Lemma 1. *There is a $FO[BIT]$–formula $Block(L, x, i, y)$ which holds iff $x < 2^L$, and the first block of length L in x is the same as the i^{th} block of length L in y.*

Proof. $Block(L, x, i, y) :\equiv \forall L' \geq L \neg BIT(L', x) \wedge \forall L' < L \left(BIT(L', x) \leftrightarrow BIT(iL + L', y) \right)$. □

In the following development, we will choose the block length L according to the situation as a rational power of $\lg N$. Since $\lg N$, the length of the binary expansion of the number N, is m iff $\exists z \leq N\, BIT(m,z) \wedge \neg\exists z \leq N\, BIT(m{+}1,z)$, we can define $(\lg N)^a$ by a $FO[BIT]$–formula, for every positive rational number $a \leq 1$.

Using blocks of appropriate length we can encode a small number of small numbers into one number.

Lemma 2. *Let a, b be positive rational numbers such that $a{+}b \leq 1$, and let $M = (\lg N)^a$, $B = 2^{(\lg N)^b}$. Then there is a $FO[BIT,P]$–formula $Count_1^{a,b}(q,P)$ which holds iff the predicate P is satisfied for exactly q numbers from $\{0,\ldots,B-1\}$ and $q \leq M$.*

Proof. Choosing $L = (\lg N)^b$, we can write every number $< B$ with L bits. Let $Count_1^{a,b}(q,P) :\equiv$

$$q \leq M \;\wedge\; \exists y \left(\forall i, i'{<}q\, \forall x \left[\big(Block(L,x,i,y) \wedge Block(L,x,i',y)\big) \;\rightarrow\; \big(x{<}B \wedge i = i'\big) \right] \wedge \right.$$

$$\left. \wedge\, \forall x < B\; [P(x) \leftrightarrow \exists i{<}q\, Block(L,x,i,y)] \right).$$

The formula asserts the existence of a number y the first q blocks of which are pairwise distinct, such that these q blocks contain precisely those members of P which are smaller than B. Since $qL \leq ML = (\lg N)^{a+b} \leq \lg N$, all q elements fit into one number. □

With this formula we can now count elements of P which are smaller than B up to threshold M. In the next step, we improve this threshold to M^s, for arbritrary s.

Lemma 3. *Let a, b, M, B be as in Lemma 2. For every s there is a $FO[BIT,P]$–formula $Count_s^{a,b}(q,P)$ which holds iff the predicate P is satisfied for exactly q numbers from $\{0,\ldots,B-1\}$ and $q \leq M^s$.*

Proof. The proof is by induction on s. Lemma 2 provides the base case $s = 1$.

For $s{+}1$, the formula $Count_{s+1}^{a,b}(q,P)$ expresses that the interval $[0,B)$ can be partitioned into at most M parts, each of which contains an appropriate number (up to M^s) of elements of P. Again, we let $L = (\lg N)^b$.

$Count_{s+1}^{a,b}(q,P)$ uses $Count_s^{a,b}$ with P replaced by the predicate $P_{[x,x')}$ which holds for z iff $x \leq z{<}x' \wedge P(z)$.

$$Count_{s+1}^{a,b}(q,P) :\equiv \Big(q \leq M^s \wedge Count_s^{a,b}(q,P)\Big)$$

$$\vee \left[M^s{<}q \;\leq\; M^{s+1} \;\wedge\; \exists q_0{\leq}M^s\, \exists q_1{<}M \left(q{=}q_0{+}q_1M^s \;\wedge\; \exists z \left[Block(L,0,0,z) \;\wedge \right.\right.\right.$$

$$\Big(\forall i{<}q_1\, \forall x, x'$$

$$\Big[\big(Block(L,x,i,z) \wedge Block(L,x',i{+}1,z)\big) \rightarrow \big(x{<}x' \;\wedge\; Count_s^{a,b}(M^s, P_{[x,x')})\big) \Big] \Big) \wedge$$

$$\left.\left.\left. \wedge\, \forall x\, \big(Block(L,x,q_1,z) \rightarrow Count_s^{a,b}(q_0, P_{[x,B)})\big) \right] \right) \right].$$

□

Let us denote by $|P|$ the number of elements of P. The proof is now completed by mapping all elements of our (small) set P 1-1 into a small initial segment. For this we choose $a = 1/3$ and $b = 2/3$, thus $L = (\lg N)^{2/3}$, $M = (\lg N)^{1/3}$ and proceed in two steps: First, we count all those 0-1–strings of length L which occur as a block in some element of P. Note that, if $|P| \leq M^s$, then there can be at most $\frac{\lg N}{L} M^s = M^{s+1}$ many such blocks. In the second step, replacing each of these blocks by its number in this count, we can represent every element of P by a sequence of very short 0-1–strings, i.e. by one small number.

Lemma 4. *For every s there is a $FO[BIT, P]$–formula $Num_s(L, q, x, P)$ which holds iff $q \leq M^{s+1}$ and x is the q^{th} smallest number which occurs as a block of length L in some member of P.*

Proof. We first define a formula $Proj(L, x, P)$ which holds iff x occurs in some element of P.

$Proj(L, x, P) :\equiv \exists i \exists y \left[P(y) \wedge Block(L, x, i, y)\right]$. Now Num_s can be defined as

$Num_s(L, q, x, P) :\equiv q \leq M^{s+1} \wedge Proj(L, x, P) \wedge Count^{a,b}_{s+1}(q, \varphi)$,
where $\varphi(z) :\equiv z \leq x \wedge Proj(L, z, P)$. □

Now we can prove the theorem.

Theorem 3. *For every k there is a $FO[BIT, P]$–formula $Size_k(q, P)$ which holds iff $q \leq (\lg N)^k$ and q is the number of elements of P.*

Proof. We let $s := 3k$ and write a formula $Repr(x, y, P) :\equiv$
$x < 2^{M \lg(M^{s+1})} \forall i, x', y' \left(Block(L, y', i, y) \wedge Block(\lg(M^{s+1}), x', i, x) \rightarrow Num_s(L, x', y', P)\right)$.
This formula asserts that x represents the element y of P by the sequence of the numbers of its blocks. Since we require x to be smaller than $2^{M \lg(M^{s+1})}$, only the first M blocks of x of length $\lg(M^{s+1})$ can be nonzero. Thus different numbers x represent different elements y, i.e., the representation is unique. Furthermore, our choice of parameters L, M ensures that all relevant elements x are smaller than $B = 2^{(\lg N)^{2/3}}$, say (for N big enough). Therefore, using Lemma 3, we can count those x which represent elements of P.

$Size_k(q, P) :\equiv q \leq M^{3k} \wedge Count_{3k}(q, \psi)$, where $\psi(z) :\equiv \exists y\, Repr(z, y, P)$.
This formula counts correctly, if $|P| \leq M^{3k} = (\lg N)^k$. □

Note that $FO[+]$-formulae can only count up to a constant (cf. [Lyn82]).

3.2 An application

In the last step of the proof of Theorem 3, we count all those numbers x which represent elements of P. We can interpret this final counting act as a bijection of P onto the initial segment of size $|P|$. Thus, our method allows us to define a bijection from any set P of polylogarithmic size to the set $\{0, \ldots, |P| - 1\}$.

Corollary 1. *For every k there is a $FO[BIT, P]$–formula $bij_k(x, y)$ which holds iff y is the x^{th} element of P, provided $|P| \leq (\lg N)^k$.*

Proof. $bij_k(x,y) :\equiv P(y) \wedge Size_k(x, P_{<y})$, where $P_{<y}(z) \iff P(z) \wedge z < y$. □

In particular, if $|P| \leq \lg N$, this enables us to represent subsets of P by 0–1 vectors of length $\lg N$, i.e., by numbers $< N$. Thus we can replace quantification over subsets of P by first–order quantification as follows. The formula $\exists X \subseteq P\, \varphi$ can be replaced by $\exists x\, \tilde{\varphi}$ where $\tilde{\varphi}$ is obtained from φ by substituting for all occurences of $y \in X$ in φ the formula $\exists z\, bij_1(z,y) \wedge BIT(z,x)$. Here, the set X is represented by a 0-1–string with a 1 in position q if the q^{th} element of P is in X.

This improves a result of Grohe and Hella (personal communication), who showed the same for $|P| = O(\lg \lg N)$. They used this in an argument to show that BIT–invariant $FO[BIT, \tau]$–definable properties are not Gaifman local.

In the remainder of this section, we will present their argument, using Corollary 1. First let us give a brief description of the problem.

It is well–known that every FO–formula φ is Gaifman local (see [Gai82]), i.e., whether or not a point in a structure satisfies φ only depends on its neighbourhood of some diameter d_φ, where d_φ only depends on the formula, not on the structure. Recently, some authors started to investigate other logics, in particular extensions of FO, with respect to this property. Whereas order–invariant $FO[<]$–formulae were shown to be local (see [GS98]), the following construction, essentially due to Grohe and Hella, shows that this is not generally true for BIT–invariant $FO[BIT]$–formulae.

Let τ be a signature which contains at least one binary relation symbol. We call a $FO[\tau, BIT]$–formula *BIT–invariant* if the following holds for any τ–structure $\mathcal{A}$: Let $<_1, <_2$ be two different linear orders on the universe of $\mathcal{A}$, and let BIT_1, BIT_2 be the corresponding BIT–relations. Then for every $a \in \mathcal{A}$, $\langle \mathcal{A}, BIT_1 \rangle \models \varphi(a) \iff \langle \mathcal{A}, BIT_2 \rangle \models \varphi(a)$.

Corollary 2. *Let τ be a signature which contains at least one binary relation symbol. There is a BIT–invariant $FO[\tau, BIT]$–formula $\varphi(x)$ such that for every d there are a τ–structure $\mathcal{A}$, a BIT–relation BIT on $\mathcal{A}$ and elements $a, b \in \mathcal{A}$, such that the d–neighbourhoods of a and b in $\mathcal{A}$ are isomorphic, but $\langle \mathcal{A}, BIT \rangle \models \varphi(a)$ and $\langle \mathcal{A}, BIT \rangle \not\models \varphi(b)$.*

Proof. Let $\Phi(x)$ be a monadic–second order formula which expresses that x lies on a circle, e.g., $\Phi(x) \equiv \exists C\, C(x) \wedge \forall u \in C\ deg_C(u)=2$. Here $deg_C(u)=2$ is a formula which expresses that u has precisely two neighbours in C. Of course, relativizing the second–order quantifier in Φ to subsets of the set $\{v \,/\, \exists w\, Evw\}$ of non–isolated vertices does not change the semantics of Φ. Let $N > 2^{4d+2}$, and let $\mathcal{A}$ be the graph on $\{0, \ldots, N-1\}$ which consists of a circle on the first $2d+1$ elements, a path on the next $2d+1$ elements, and in which all other vertices are isolated. Then the set of non–isolated vertices in $\mathcal{A}$ is of size $\lg N$, and choosing $a = d$ and $b = 3d+1$, we see that the d–neighbourhoods of a and b are isomorphic and that $\mathcal{A} \models \Phi(a)$, but $\mathcal{A} \not\models \Phi(b)$. Thus, if we replace Φ by the $FO[\tau, BIT]$–formula $\tilde{\Phi}$, as described above, then $\tilde{\Phi}$ is BIT–invariant, but $\langle \mathcal{A}, BIT \rangle \models \tilde{\Phi}(a)$, and $\langle \mathcal{A}, BIT \rangle \not\models \tilde{\Phi}(b)$, for any choice of BIT as a BIT–relation. □

4 Counting in the Linear Hierarchy

In this section, we present an elementary proof of Theorem 2. As we have said before, rudimentary predicates admit a lot of alternative characterizations. We will use this fact and present the proof in the language of alternating algorithms. We first establish the following simpler result.

Lemma 5. *Let $P(N, \boldsymbol{u})$ be a rudimentary relation. Then, for every fixed rational $\epsilon < 1$ and every fixed integer k, the counting relation*

$$C_\epsilon P(N, y, \boldsymbol{u}) \equiv \big(\ y = \sharp\{x < N;\ P(x, \boldsymbol{u})\ \wedge\ \lg x \leq (\lg N)^\epsilon\ \wedge\ y \leq (\lg N)^k\big)$$

is also rudimentary.

We will not mention the $\boldsymbol{u}$ variables in the proof. Below, if l is an integer and α is a rational number then l^α stands for $\lceil l^\alpha \rceil$. Also $P_{[\alpha,\beta)}(x)$ stands for $P(x) \wedge \alpha \leq x < \beta$.

Proof. Let $l = \lg N$, ϵ be a rational < 1 and k be a fixed integer. There exists h s.t. $l^k \leq l^{h(1-\epsilon)}$. The proof is by induction on h on powers $l^{h(1-\epsilon)}$ of l. The base case ($h = 1$) is handled by the following algorithm:

- Guess $x_1, x_2, \ldots, x_y$ of length less than l^ϵ such that $0 \leq x_1 < x_2 < \ldots < x_y < N$.
  The total number of chosen bits is less than $yl^\epsilon \leq \lceil l^{(1-\epsilon)} \rceil l^\epsilon \leq O(l)$. Let $x_0 = 0$, $x_{y+1} = N$ and verify that:
- For all $i \in [0, y+1)$,
  `a:` if $i \neq 0$ then $P(x_i)$,
  `b:` for all $x \in (x_i, x_{i+1})$, $\lg x \leq l^\epsilon \rightarrow \neg P(x)$.

This algorithm simply checks that the x_i's are the only integers of size bounded by l^ϵ such that $P(x_i)$ holds. It rejects if for every choice of $x_1, \ldots, x_y$ the last item is not verified. Basic arithmetical manipulations (building l^ϵ, multiplying integers ...) can be done easily in linear time with a constant number of alternations. Then, compared to the alternation depth of the algorithm deciding P, a constant number of additional levels of alternation are required.

We now show how to count up to $l^{h(1-\epsilon)}$, knowing how to do it up to $l^{(h-1)(1-\epsilon)}$. The algorithm is:

- Guess $y_0 \leq l^{1-\epsilon}$ and $y_1 \leq l^{(h-1)(1-\epsilon)}$ s.t. $y = y_0 l^{(h-1)(1-\epsilon)} + y_1$. Guess med of size bounded by l^ϵ.
- Verify that $C_\epsilon P_{[med,N)}(N, y_1)$ holds (i.e. that $y_1 = \#\{med \leq x < N;\ P(x)\}$).
- Guess $x_0, \ldots, x_{y_0}$ with $x_0 = 0$, $x_{y_0} = med$ and $x_i < x_{i+1}$ for $i > 0$. As y_0 is bounded by $l^{1-\epsilon}$ the total number of guessed bits is bounded by some $O(l)$.
- For all $i \in [0, y_0)$ verify that, $C_\epsilon P_{[x_i, x_{i+1})}(N, l^{(h-1)(1-\epsilon)})$ holds (i.e. $l^{(h-1)(1-\epsilon)} = \#\{x_i \leq x < x_{i+1};\ P(x)\}$).

The induction hypothesis is used to perform the verification steps. Here it is simply checked that there exist exactly y_1 integers (of small size) satisfying P between *med* and x and y_0 blocks of $l^{(h-1)(1-\epsilon)}$ integers verifying P between 0 and *med*. Again, it is easily seen that a constant number of supplementary alternations are sufficient in the induction step.

We are now ready to give the proof of the main result

Proof of Theorem 2. Let $\epsilon < 1$, $l = \lg N$ and $c = \lceil l^{1-\epsilon} \rceil$. Every integer r of length bounded by l can be represented by a vector $(r_1, \ldots, r_c)$ where each r_i is of length bounded by l^ϵ. It can easily be shown that the predicate $\mathsf{Proj}(r, \alpha)$, true when $\bigvee_{i=1}^{c} (\alpha = r_i)$, is rudimentary.

Let $\mathsf{Count\text{-}Coord}(N, \alpha, \beta)$ be the predicate that holds when :

$$\begin{aligned} \beta = \#\{\alpha' \leq \alpha;\ \exists r \leq N\ \mathsf{Proj}(r, \alpha')\ \ \&\ P(r)\}\ \ \& \\ \exists r \leq N\ \mathsf{Proj}(r, \alpha)\ \&\ P(r)\ \ \& \\ \lg \alpha \leq l^\epsilon \end{aligned}$$

$\mathsf{Count\text{-}Coord}(N, \alpha, \beta)$ asserts that α is the βth number that appears as a coordinate (in some place) in the vector representation of some "good" integer less than N.

If the number of good integers less than N is bounded by some l^k then β is bounded by l^{k+1}. From Lemma 5 (and the fact that $\lg \alpha' \leq \lg \alpha \leq l^\epsilon$), this implies that $\mathsf{Count\text{-}Coord}(N, \alpha, \beta)$ is rudimentary. This also implies that every "good" integer r less than N can be represented by the following s, of size $O(c \lg l)$:

$$(s_1, \ldots, s_c)$$

where each s_i satisfies $\mathsf{Count\text{-}Coord}(N, r_i, s_i)$ i.e. s_i is the rank of r_i in the sense defined above. Every such s is called a *good code* and satisfies the predicate $\mathsf{Good\text{-}Code}(N, s) \equiv \exists r < N \forall i \leq c\ P(r) \wedge \mathsf{Count\text{-}Coord}(N, r_i, s_i)$.

Finally, since *good codes* are few and of small size (in $O(c \lg l)$), from Lemma 5 they can be enumerated. Then, the proof is achieved by verifying that there exist exactly y numbers of size bounded by some $O(c \lg l)$ that represent *good* integers (*y good codes*). This is done by the algorithm below:

Guess s of size bounded by $O(c \lg l)$ and verify that:

- $\mathsf{Good\text{-}Code}(N, s)$
- $y = \#\{s' \leq s : \mathsf{Good\text{-}Code}(N, s')\}$
- for all $s' > s$, $\neg \mathsf{Good\text{-}Code}(N, s')$. □

Acknowledgments

We would like to thank Lauri Hella and Martin Grohe for allowing us to include Corollary 2 into this paper. Thanks go also to Thomas Wilke for pointing out a simplification in the proof of Corollary 1.

References

[ABO84] M. Ajtai and M. Ben-Or. A Theorem on Probabilistic Constant Depth Computations. *Proc. 16th ACM STOC*, pages 471–474, 1984.

[Ajt83] M. Ajtai. Σ^1_1 Formulae on Finite Structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[DDLW98] Anuj Dawar, Kees Doets, Steven Lindell, and Scott Weinstein. Elementary properties of the finite ranks. *Math. Log. Q.*, 44:349–353, 1998.

[DGS86] L. Denenberg, Y. Gurevich, and S. Shelah. Definability by Constant-Depth Polynomial-Size Circuits. *Information and Control*, 70:216–240, 1986.

[FKPS85] R. Fagin, M. Klawe, N. Pippenger, and L. Stockmeyer. Bounded-Depth, Polynomial-Size Circuits for Symmetric Functions. *Theor. Comput. Sci.*, 36:239–250, 1985.

[FSS81] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. In *FOCS*, pages 260–270. IEEE, 1981.

[Gai82] H. Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North Holland, 1982.

[GS98] Martin Grohe and Thomas Schwentick. Locality of order-invariant first-order formulas. In Lubos Brim, Jozef Gruska, and Jirí Zlatuska, editors, *MFCS*, volume 1450 of *Lecture Notes in Computer Science*, pages 437–445. Springer, 1998.

[HP93] Petr Hájek and Pavel Pudlák. *Metamathematics of first-order arithmetic*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1993.

[Imm87] Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

[Imm99] N. Immerman. *Descriptive Complexity*. Springer, New York, 1999.

[Lyn82] James F. Lynch. On sets of relations definable by addition. *J. Symb. Log.*, 47(3):659–668, 1982.

[MO97] Malika More and Frédéric Olive. Rudimentary languages and second order logic. *Math. Log. Q*, 43:419–426, 1997.

[Nep70] V. A. Nepomnjaščiĭ. Rudimentary predicates and Turing computations. *Dokl. Akad. Nauk SSSR*, 195:282–284, 1970.

[PW87] J. Paris and A. Wilkie. Counting Δ_0 sets. *Fund. Math.*, 127(1):67–76, 1987.

[Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[Wra78] Celia Wrathall. Rudimentary predicates and relative computation. *SIAM J. Comput.*, 7(2):194–209, 1978.