THE COMPLEXITY OF ALGORITHMIC PROBLEMS ON SUCCINCT INSTANCES*

José L. Balcázar, Antoni Lozano, and Jacobo Torán

Dep. Llenguatges i Sistemes Informàtics
Univ. Politècnica de Catalunya (ed. FIB)
08028 Barcelona, Spain

balqui@lsi.upc.es, lozano@lsi.upc.es, jacobo@lsi.upc.es

Abstract: Highly regular combinatorial objects can be represented advantageously by some kind of description shorter than their full standard encoding. For instance, graphs exhibiting enough regularities can be described using encodings substantially shorter than the full adjacency matrix. A natural scheme for such succinct representations is by means of boolean circuits computing, as a boolean function, the values of individual bits of the binary encoding of the object. The complexity of many algorithmic problems changes drastically when this succinct representation is used to present the input. Two powerful lemmas quantifying exactly this increase of complexity are presented. These are applied to show that previous results in the area can be interpreted as sufficient conditions for completeness in the logarithmic time and polynomial time counting hierarchies.

1. INTRODUCTION

One of the main goals of complexity theory is the classification of computational problems in complexity classes according to the amount of resources needed to solve them. Inputs to the algorithms are usually binary encodings of certain combinatorial objects, such as graphs. Actually, in order to use efficient combinatorial algorithms to solve problems by computer, such combinatorial structures must be found or imposed on the data to the programs.

* This research was partially supported by the ESPRIT-II Basic Research Actions Program of the European Community under contract no. 3075 (project ALCOM).

Consider combinatorial objects to be treated by computer programs. The case may arise that these objects turn out to be highly regular. Representing such regular objects by means of data structures that take advantage of these regularities is a very natural idea, since memory space in computers should be employed efficiently and therefore it is good to encode the objects in a more succinct way.

For instance, data structures for sparse matrices, with a small number of nonzero entries, have been studied since long time. Actually, such data matrices appear in many applications of combinatorial optimization problems. Several other possibilities have been proposed to obtain representations of regular graphs smaller than the adjacency matrix ([6], [28], [29]).

Here we continue a study initiated in [6], where the adjacency matrix of a graph is represented by a hopefully small boolean circuit which, on input the binary representations of i and j, computes the (i, j) entry of the adjacency matrix. We will study problems not only on graphs, but also on numbers, strings, sets, and boolean functions. We are interested in using boolean circuits to describe the inputs to algorithms for various decisional problems. On the one hand, since the size of the input may be much smaller, the complexity of the algorithms must be expected to grow. On the other hand, since only very regular objects can be described by circuits with substantial degree of succinctness, one might hope that some trade-off exists, making the problem easier for compactly described instances due to the regularity of the represented object.

In [6] and subsequently in [20] these hopes are destroyed for many graph problems. Indeed, it is shown in [6] that checking quite simple graph properties, such as nonemptiness of the set of edges, become *NP*-hard if the input is given by a circuit. A sufficient condition for *NP*-hardness of succinct versions of problems is given; we will discuss this further later on. Many more complex problems jump to much higher complexity classes. This work was extended in [20] by proving that sets that are *NP*-complete for certain reducibility (known as "projection") become *NEXPTIME*-complete when the input is given by a circuit representation. This last reference states that similar techniques can be applied to projection-complete sets in other complexity classes, such as *P* or *NL*, and asserts that from such a construction for *NL* other lower bounds of [6] can be improved to optimal.

Our aim in this paper is to show how to distill from the proof of the main result of [20] some uniform, powerful lemmas, with simpler proofs, which can be used to, first, obtain uniformly all the results known about the complexity of succinct versions of particular problems; second, to add many more to the list, and third, to show how the sufficient condition for *NP*-hardness of the succinct version of a problem, given in [6], is essentially a sufficient condition for the hardness of the standard problem with respect to a much less powerful class. To do this, we will introduce a form of reducibility more general than that of [20], and will show that their use of Cook's formula, restricted to a case of low Kolmogorov complexity, is actually unnecesary and can be replaced by a simpler encoding characterized also by its low information contents.

Our main lemma is completely independent of the class on which the computational problems lie, since its statement regards just decisional problems and their succinct versions. This presentation is both easier and better, since the independence of complexity classes allows us to carry over the result to classes with surprisingly low computational power.

Actually, our results can be applied to find complete sets for the lower classes of the logarithmic time and the polynomial time counting hierarchies, as well as for arbitrary levels of some subhierarchies. In this way we obtain precise classifications of the complexity of many problems about finite functions, both under the standard and under the succinct encoding.

In [20] it is pointed out that a proof similar to the one they give applies to problems on directed graphs such as accesibility. However, to apply it to undirected accessibility, or to other problems on undirected graphs such as connectivity, bipartiteness, or acyclicity, some difficult results must be used which prove that acceptance of deterministic logarithmic space machines can be reduced to undirected accessibility [16], by means of the class SL (symmetric logarithmic space). We give here a different proof of these results, which we consider very instructive since it pushes further a technique of [6] which the authors did not expect to work. We apply it also to other problems, namely planarity, and using results of Reif [21] we settle the particular case of bounded degree graphs.

This paper includes and extends results announced in [18] and [26], which in turn were based on parts of the M. Sc. Thesis of the second author and the Ph. D. dissertation of the third, both under the advice of the first.

2. COMPLEXITY CLASSES AND REDUCIBILITIES

We consider inputs to decisional problems encoded as words over the two letter alphabet $\Gamma = \{0,1\}$. In particular the standard encoding of a binary string is itself; the standard encoding of a positive integer is its binary description, in principle with no leading zeros (although leading zeros may be accepted for padding purposes); that of a finite set, its characteristic function, usually as a subset of Γ^n; that of a directed graph, its adjacency matrix; and that of a boolean function, its truth table (generalized to multiple outputs if necessary). Observe that all those encodings can be seen simply as binary words.

Regarding undirected graphs, for technical reasons we allow as encoding of an undirected graph the encoding of any of the directed graphs that can be obtained from it by directing the edges. Thus, edge (i,j) is in an undirected graph if and only if either of the entries (i,j) or (j,i) (or both) are 1 in the adjacency matrix. The property we need of this representation is that adding an edge can be done by changing a single bit of the encoding.

Given a binary string x, we denote by $[x]$ the integer represented in binary by x; sometimes we will explicitly prefix x with a 1 beforehand to avoid problems with leading zeros. The length (i.e. number of bits) of a binary string x is denoted $|x|$, and the i-th bit of x is denoted x_i. The cardinality of a finite set B is denoted $||B||$. We identify each decisional problem with the set of encodings of inputs on which the answer is YES.

We mention in the text several complexity classes, i.e. classes of sets that can be decided within a given resource bound by a sequential computation model such as, e.g., multitape Turing machines. Results regarding complexity classes may be found, among others, in [1] and [7]. These complexity classes are: deterministic logarithmic space, denoted L, nondeterministic logarithmic space, denoted NL, deterministic polynomial time, denoted P, nondeterministic polynomial time, denoted NP, polynomial space, for which deterministic and nondeterministic classes coincide and therefore is denoted just $PSPACE$, deterministic exponential (i.e. $O(2^{n^k})$ for constant k) time, denoted $EXPTIME$, and nondeterministic exponential time, denoted $NEXPTIME$. We denote by FP the set of functions computable in polynomial time.

We will work also with some complexity classes of very low computational power, defined by machines working in logarithmic time. The device for reading the input usually provided to a standard Turing machine is sequential: in logarithmic time, only the first $O(\log n)$ bits of the input could be read. In order that all bits of the input be equally relevant, we will use a known variant of Turing machine with direct access to the bits of the input. The machine will include the following elements:

- an input tape;
- a fixed number of standard work tapes;
- a special tape to point to a bit of the input, which may be subsequently read in;
- a special tape on which the symbol just read from the input appears written;
- a "read" state.

The machine is otherwise standard. It reads its input in the following way: the machine can write on the pointer tape the number of a position i of the input tape; whenever the "read" state is entered, in one computation step, the machine gets in the second special tape the contents of the i-th position of the input tape. If the input has length less than i, then the machine does not get anything. The previous contents of the second special tape is overwritten, but the contents of the pointer tape and the position of its head remain untouched.

Under a linear or larger than linear time bound, these machines can be simulated by standard Turing machines within roughly the same resource bounds. The interest of indirect access machines arises when we consider sublinear time bounds. In particular, we will consider indirect access machines with computation time bounded by $O(\log n)$.

1. *Example.* Let us show how an indirect access machine working in $O(\log n)$ steps can compute the length of its input; this fact will be used later on. This is done by using the following binary search technique: first, probe positions 2^i of the input tape, increasingly, until the first empty cell is found; every probe can be done in constant time since in order to change the contents of the pointer tape from 2^i to 2^{i+1} only one (or two) bits must change. Once an integer k is found such that $2^k \leq |x| < 2^{k+1}$, perform a binary search to find the last input position; again each probe requires only constant time since at most one bit of the pointer tape has to change. The whole procedure requires a logarithmic number of steps.

We will denote by LT the class of languages accepted by deterministic indirect access Turing machines within a computation time bounded by $O(\log n)$, and similarly we denote FLT the class of functions computable by such machines in $O(\log n)$ time.

The concept allowing us to compare the difficulty of different problems is reducibility. Intuitively, a problem A is reducible to a problem B if an efficient algorithm for solving B can be transformed into an efficient algorithm for solving A.

We will use two reducibilities. The first is the standard polynomial time m-reducibility, defined as follows: problem A is polynomial time m-reducible to problem

B, denoted $A \leq_m^P B$, if and only if there is a polynomial-time computable function f such that $x \in A \iff f(x) \in B$ for every string x. We will call this simply m-reducibility, omitting "polynomial time."

The second reducibility is similar, but under a more strict resource bound: logarithmic time m-reducibility, which we will abbreviate to LT-reducibility and denote as $\leq_m^{LT}$. We will need such a fine reducibility to work at the level of the logarithmic-time counting hierarchy. The first approximation is that $A \leq_m^{LT} B$ if and only if there is a logarithmic-time computable function f such that $x \in A \iff f(x) \in B$ for every x. However, this is inappropriate since in logarithmic time big problem instances could only be reduced to small problem instances. We avoid these problems by using a definition of reducibility which translates "locally" parts of problem instances in logarithmic time.

Thus, our condition on the reduction function f for the definition of LT-reducibility will be that the following function φ be computable in logarithmic time: for $x \in \Gamma^*$ and $i \in \mathbb{N}$, $\varphi(x,i)$ is $f(x)_i$ (the i-th bit of $f(x)$) if $i \leq |f(x)|$, and is undefined otherwise.

Previous work on succinct instances used projection reducibility [20], [18]. This is a very weak reducibility, a version of the nonuniform projection reducibility of [23], and is characterized by the fact that each nonconstant bit of the output coincides with a bit of the input (or its negation), found in a fixed position which depends only on the given position in the output. We will not use this reducibility here.

Given a reducibility and a complexity class, a problem is *hard* for the class under the reducibility if every problem in the class is reducible to it, and is *complete* for the class if it is hard for it and also belongs to it.

3. SUCCINCT INSTANCES AND THE CONVERSION LEMMA

This section proposes a different, simpler and more general way of presenting some of the arguments of [20], so that the results there, previous ones from [6] and [29], and many others then follow immediately in a unified fashion.

A succinct representation of a binary word x is a boolean circuit that on input (the binary representation of) i outputs two boolean values, one indicating whether $i \leq |x|$ and another one indicating, in that case, the i-th bit of x. The same criteria can be used to define succinct representations of integers, finite sets, graphs, or boolean functions. For instance, in the case of graphs, a succinct representation is a boolean circuit which, on input the binary representations of i and j, computes the entry (i,j) of the adjacency matrix.

The succinct version sA of any decisional problem A is: given a boolean circuit describing a word x, decide whether $x \in A$.

The succinctly represented problem is at most exponentially more difficult than the problem on standard encodings. The following observation is proved for graphs in [6], and can be stated similarly for any binary encoding.

2. *Lemma.* Let $f(n) \geq \log n$ be a nondecreasing bound, and let A be a decisional problem. Assume that $A \in DTIME(f(n))$. Then, for the succinct version sA of A, we have $sA \in DTIME(f(2^n)n^2)$.

The same holds for *NTIME*, and similarly for *DSPACE* or *NSPACE*, in which case the new space bound is simply $f(2^n)$. A later lemma will state similar facts for complexity classes in the counting hierarchies.

The same reference proves a sufficient condition for *NP*-hardness of succinct problems: if certain "critical" graphs exist for a graph property, then testing this property on succinctly represented graphs is *NP*-hard. Actually, the form in which this condition is stated in [6] is too weak for the proof to be correct: the interested reader can find in [17] counterexamples not only to the proof, but (assuming $P \neq NP$) to the statement itself. The expert in structural complexity notions will enjoy the constructions, based on the concept of bi-immunity. The corrected statement of the sufficient condition for *NP*-hardness is also given in [17].

We will see in this paper that this kind of sufficient condition, seemingly tailored to succinct representations, actually can be applied to standard encodings, as soon as the logarithmic-time complexity classes are available (theorem 11 below). Then, the results of [6] follow by combining the criticality argument with the precise relationships between standard and succinct encodings that follow from the next two lemmas.

The following lemma, which we name *Conversion Lemma*, shows how to obtain m-reducibilities among succinctly represented problems from LT-reducibilities among the standard problems. Some ideas of its proof are based on a fragment of the proof of the main theorem of [20]. The advantages are, first, that a more flexible and useful form of reducibility among the standard problems is used here (at the price of a technical complication in the proof), and second, that it is now independent of the fact that the problems belong to any particular complexity class.

3. *Lemma.* (*Conversion Lemma*) If $A \leq_m^{LT} B$ then $sA \leq_m^P sB$.

Proof. Let w be an instance of A. Assume as input an instance of sA, given by a boolean circuit C_w describing w. Observe that the size of C_w is at least $\log |w|$, which is the number of inputs, and therefore $|w| \leq 2^{|C_w|}$.

Let f be the reduction function: for all w, $w \in A \iff f(w) \in B$; and let M be the machine that, on input $\langle w, i \rangle$, computes the i-th bit of $f(w)$ in logarithmic time. There is a standard translation of Turing machines into circuits (see [1], section 5.4), appropriate for Turing machines that read directly their input tapes, yielding a circuit of size quadratic in the running time of the machine computing the same output. Each level of the circuit simulates one step of the machine.

Now if the Turing machine has indirect access, the mechanism to supply the input bits in the standard simulation does not work. Fortunately, by hypothesis we have now a circuit C_w supplying individual bits of the input w. Assume that the machine starts by reading in sequentially and storing on worktape the second argument i. This part is simulated by the circuit in the straightforward way since i is read sequentially. For the remaining part of the computation, where bits of w must be read, we can insert one copy of the circuit C_w at each level, fed by the bits in the pointer tape (which are computed by other gates of that level), so that the requested input bit is available at the output of C_w. This extended circuit C' (which is now of size at most cubic on $|C_w|$) does not need w as input anymore: simply, on input i, it produces the i-th bit of $f(w)$.

Therefore, this circuit C' is precisely what we need: a succinct description of $f(w)$, the image of w under the LT-reduction. Indeed, now C_w is in sA if and only if

w is in A, if and only if $f(w)$ is in B, if and only if C' is in sB. It is easy to see that C' can be computed in time polynomial in the size of C_w. ■

In order to prove completeness of a succinctly represented problem sA in a complexity class, we need a way of translating arbitrary problems in the class into a region where we can use a hypothesis about A, which is exponentially below. (Actually, we are addressing now the problem solved in a more complex way in another fragment of the main proof of [20]).

A possibility could be using tally encodings, which will be of much lower complexity due to the exponential blow-up of the input size; but this requires to check that all bits are equal, and this cannot be done in logarithmic time. Yet we will encode problems in a form similar to tally sets, and with similar properties. For a problem A, define long(A) as follows: $x \in \text{long}(A)$ if and only if the binary expression of the length of x is in $1A$. (The 1 prefixing all words here avoids the technical difficulty of the leading zeros.) Thus, for each length n, either all the words of that length are in long(A), if the binary expression of n is in $1A$, or all the words of that length are out of long(A), otherwise. Observe that

$$A \in DTIME(f(2^n)) \iff \text{long}(A) \in DTIME(f(n))$$

and similarly for other complexity classes. We prove next that the succinct version of long(A) jumps up again at least to the difficulty of A.

4. *Lemma.* For every A, $A \leq_m^P s(\text{long}(A))$.

Proof. Since A trivially reduces to $1A$ and the reducibility is transitive, we exhibit a reduction from $1A$ to $s(\text{long}(A))$. Recall that $[x]$ is the integer denoted by x in binary notation. Given x starting with 1, construct a circuit C representing the word $0^{[x]}$, which outputs zero for all inputs up to $[x]$. The time required to construct this circuit is easily seen to be polynomial in $|x|$, and by the definitions we have $x \in 1A \iff 0^{[x]} \in \text{long}(A) \iff C \in s(\text{long}(A))$. ■

Now we can state and easily prove the following result, which will be readily applied. The proof is simple since the essence of it has been captured by the Conversion Lemma.

5. *Theorem.* Let C_1 and C_2 be arbitrary complexity classes such that for every $A \in C_1$, $\text{long}(A) \in C_2$. Then, for every B, if B is hard for C_2 under LT-reducibility, then sB is hard for C_1 under m-reducibility.

Proof. Assume that B is hard for C_2 under LT-reducibility, and let A be an arbitrary set $A \in C_1$. The hypothesis imply $\text{long}(A) \in C_2$ and therefore $\text{long}(A) \leq_m^{LT} B$. By the Conversion Lemma, $s(\text{long}(A)) \leq_m^P sB$. Since by lemma 4 $A \leq_m^P s(\text{long}(A))$, by transitivity, $A \leq_m^P sB$ and thus sB is m-hard for C_1. ■

As an example of application, let us prove now that the results fully proved in [20] can be obtained as immediate corollary: since Hamiltonian path, k-colorability, and many other NP-complete sets can be proved complete via LT-reductions [7], we have:

6. *Corollary.* The succinct versions of Hamiltonian path, k-colorability, etc. are m-complete in *NEXPTIME*.

Proof. Membership follows from lemma 4, and hardness from theorem 5. ∎

Subsequent sections will describe further applications of this set of lemmas, capturing in very clear ways the essentials of the optimal results about *NP*-completeness from [6] and many other results.

4. THE COUNTING HIERARCHIES

Probably the best known complexity classes are P and *NP*. Also well-known is the polynomial-time hierarchy *PH* [24], a natural generalization of the class *NP*. Nevertheless there are many natural computational problems whose complexity cannot be modelized in terms of existential or universal quantifiers; sometimes this complexity is captured by other complexity classes, more adapted to the idea of *counting*.

Wagner [29] defines the counting hierarchy *CH* in a similar way as the polynomial-time hierarchy. The counting hierarchy turns out to be a very useful tool to express the complexity of many natural problems. It contains the polynomial-time hierarchy and is included in *PSPACE*. Wagner shows that every level of *CH* has complete problems and proves some other properties.

In this section we will define the polynomial-time counting hierarchy *CH* and the logarithmic-time counting hierarchy *LCH*. The classes in these hierarchies are characterized by polynomially bounded quantification, resp. logarithmically bounded quantification, over a predicate computed in polynomial, resp. logarithmic time.

The polynomial counting quantifier $\mathbf{C}$ is defined in the following way: for a function $f : \Gamma^\star \mapsto \mathbb{N}$, $f \in FP$, a polynomial p and a binary predicate P,

$$\mathbf{C}^p_{f(x)} y : P(x,y) \iff ||\{y : |y| \leq p(|x|) \text{ and } P(x,y)\}|| \geq f(x).$$

If K is a language class, for any set A, $A \in \mathbf{C}K$ if there is a function f in *FP*, a polynomial p, and a language $B \in K$ such that for any $x \in \Gamma^\star$

$$x \in A \iff \mathbf{C}^p_{f(x)} y : \langle x, y \rangle \in B$$

The polynomial counting hierarchy *CH* is the smallest family of language classes satisfying:

i/ $P \in CH$;

ii/ if $K \in CH$ then $\exists^p K$, $\forall^p K$ and $\mathbf{C}^p K$ belong to *CH*.

For simplicity, $\mathbf{C}$ will denote the class $\mathbf{C}P$, and the context will make clear when we talk about a quantifier and when about a language class. Also, we will drop the superscript p from all the quantifiers.

We should point out here the following properties:

7. *Proposition.*

a/ $CH \subseteq PSPACE$

b/ $PH \subseteq CH$

c/ For every class K in CH, $\exists K \cup \forall K \subseteq \mathbf{C}K \subseteq \exists \mathbf{C}K \cap \forall \mathbf{C}K$.

d/ Every class in CH is closed under m-reducibility.

e/ Every class in CH has complete problems with respect to the m-reducibility.

Similarly, we define now the logarithmic-time counting hierarchy. The logarithmic existential, universal, and counting quantifiers are defined in the following way: for a function $f : \Gamma^\star \mapsto \mathbb{N}$ computable in logarithmic time, a constant c and a binary predicate P,

$$\exists^{lc} y : P(x,y) \iff \bigvee_y |y| \leq c\lceil \log|x| \rceil \text{ and } P(x,y)$$

$$\forall^{lc} y : P(x,y) \iff \bigwedge_y |y| \leq c\lceil \log|x| \rceil \text{ and } P(x,y)$$

$$\mathbf{C}^{lc}_{f(x)} y : P(x,y) \iff ||\{y : |y| \leq c\lceil \log|x| \rceil \text{ and } P(x,y)\}|| \geq f(x)$$

If K is a language class, for any set A, $A \in \exists^l K$ if there is a language $B \in K$ and a constant c such that for any $x \in \Gamma^\star$

$$x \in A \iff \exists^{lc} y : \langle x,y \rangle \in B$$

and analogously for $A \in \forall^l K$. $A \in \mathbf{C}^l K$ if there is a language $B \in K$, a constant c and a function f computable in logarithmic time such that for any $x \in \Gamma^\star$

$$x \in A \iff \mathbf{C}^{lc}_{f(x)} y : \langle x,y \rangle \in B$$

The logarithmic-time counting hierarchy LCH is the smallest family of language classes satisfying

i/ $LT \in LCH$;

ii/ if $K \in LCH$ then $\exists^l K, \forall^l K$ and $\mathbf{C}^l K$ belong to LCH.

In [2], the alternating log-time hierarchy LH is defined using log-time indirect access Turing machines that alternate between existential and universal states. This hierarchy coincides with the subfamily of LCH defined using only $\exists^l$ and $\forall^l$ quantifiers, and is known to be proper [22]. Additionally, in [5] it is shown that $\mathbf{C}^l LT$ is not included in LH, and in [27] it is proved that the inclusions $\mathbf{C}^l LT \subset \exists^l \mathbf{C}^l LT$ and $\mathbf{C}^l LT \subset \forall^l \mathbf{C}^l LT$ are strict.

By analogy with the polynomial-time hierarchy, and following common usage, we will call $\Sigma^l_k = \exists^l \forall^l \exists^l \ldots Q^l LT$, the class defined by $k-1$ alternations of logarithmically bounded quantifiers $\exists$ and $\forall$, being $Q^l = \forall^l$ if k is even and $\exists^l$ if k is odd; also we will use the notation $\mathbf{C}^l_k = \mathbf{C}^l \mathbf{C}^l \ldots \mathbf{C}^l LT$ for the class defined by k logarithmically bounded counting quantifiers. We will denote sometimes $NLT = \exists^l LT$ since it corresponds to nondeterministic logarithmic time.

Several properties and inclusions of CH translate directly to LCH. Let us point out just three properties.

8. *Proposition.*

a/ $LCH \subseteq L$.

b/ For every class K in LCH, $\exists^l K \cup \forall^l K \subseteq \mathbf{C}^l K \subseteq \exists^l K \cap \forall^l K$.

c/ If two classes of the LCH coincide, so do their corresponding classes of CH.

The next decision problems are complete in $\exists^l LT$ and in $\mathbf{C}^l LT$ respectively.

9. *Example.* Nonzero string.

Let $L_1 = \{w \in \Gamma^\star : w \text{ has at least one } 1\}$.

L_1 is $\exists^l LT$-complete. Let us prove it.

$L_1 \in \exists^l LT$. Consider the language L_0 accepted by a deterministic indirect access log-time machine M that on input $\langle x, i\rangle$, $|i| = \lceil \log |x| \rceil$, accepts if and only if the i-th bit of x is a 1. It is clear that $L_0 \in LT$ and that for all $x \in \Gamma^\star$,

$$x \in L_1 \iff \bigvee_i |i| \leq \lceil \log |x| \rceil \langle x, i\rangle \in L_0$$

L_1 is hard for $\exists^l LT$ with respect to the LT-reducibility. Let $L \in \exists^l LT$

$$x \in L \iff \bigvee_i |i| \leq c\lceil \log |x| \rceil P(x, i)$$

being $P(x,i)$ a predicate in LT. Consider the function g and a logarithmic-time computable function φ computing the bits of $g(x)$ as follows: $g(x)$ will have a "1" in position i if the i-th string of length $\leq c\lceil \log |x| \rceil$ is a witness of $x \in L$, and it will have a "0" in this position otherwise. For a given x, let $n = c\lceil \log |x| \rceil$. Then for $|i| \leq n$,

$$\varphi(x,i) = \begin{cases} 1 & \text{if } P(x,i) \\ 0 & \text{if } \neg P(x,i) \end{cases}$$

The function φ can be computed in log-time since $P(x,i)$ is a log-time predicate, and $n = c\lceil \log |x| \rceil$ can be calculated by first computing $l = |x|$, as described in example 1, then computing the length of l, which is $\lceil \log |x| \rceil$, and multiplying by the constant c. It is clear that $x \in L \iff g(x) \in L_1$.

10. *Example.* Majority.

Let $L_2 = \{w \in \Gamma^\star : w \text{ has more 1's than 0's}\}$.

Let us prove that L_2 is $\mathbf{C}^l LT$-complete. Observe that $w \in L_2$ if and only if more than half of the bits of w are 1.

$L_2 \in \mathbf{C}^l LT$. Consider the LT language L_0 of the previous example. For all $x \in \Gamma^\star$

$$x \in L_2 \iff ||\{i : |i| \leq \lceil \log |x| \rceil \langle x, i\rangle \in L_0\}|| \geq \left\lfloor \frac{|x|}{2} \right\rfloor + 1$$

and this bounding function is log-time computable.

L_2 is hard for $\mathbf{C}^l LT$ with respect to LT-reducibility. Let $L \in \mathbf{C}^l LT$

$$x \in L \iff ||\{i : |i| \leq c\lceil \log |x| \rceil P(x,i)\}|| \geq f(x)$$

being $P(x,i)$ a predicate in LT and f a function in FLT. Consider again functions g and φ like in the previous example, the difference lying in that now we have to create a certain amount of 1's to change the bound f into one half. Once more, for a given x, let $n = \lceil c \log |x| \rceil$.

$$\varphi(x,i) = \begin{cases} 1 & \text{if } |i| \leq n \text{ and } P(x,i) \\ 0 & \text{if } |i| \leq n \text{ and } \neg P(x,i) \\ 0 & \text{if } n < |i| \leq n + f(x) - 1 \\ 1 & \text{if } n + f(x) - 1 < |i| \leq 2n \end{cases}$$

By the same arguments as for the reduction to L_1, φ can be computed in log-time. Observe that here the valued mapped to by the reduction contains in its second part $n - f(x) + 1$ ones, so that $g(x) \in L_2$ if and only if in the first part there are at least $f(x)$ ones. This technique can be applied in many other contexts, and transforms an arbitrary counting quantifier into one whose threshold is exactly $1/2$. Thus we will assume, whenever needed, that our counting quantifiers have threshold $1/2$, by appealing to this construction.

5. SUFFICIENT CONDITIONS FOR HARDNESS

The purpose of this section is to present a general tool for proving LT-hardness results for some classes of the logarithmic-time counting hierarchy. Essentially, such ideas appear in [6], where they are presented as sufficient conditions for succinct versions of problems being *NP*-hard or Σ_2-hard. As can be seen from our results in this section, their *NP*-hardness results can be strengthened substantially with the help of the Conversion Lemma and the logarithmic-time counting hierarchy: we will show a similar but more general sufficient condition for problems being *NLT*-hard, and several applications; then, in the next section, the Conversion Lemma will apply to show immediately that the corresponding succinct versions are *NP*-hard, encompassing all *NP*-completeness or *co-NP*-completeness results of [6] and [29] and many more cases. More importantly, in this way an explanation is found for the fact that many succinct problems are *NP*-complete.

As a general setting, we study the complexity of problems stated on boolean functions of finite domain, always of the form $f : \Gamma^n \mapsto \Gamma^m$. From now on we call these objects "finite functions". The encoding of such a function f is by 2^n blocks of m bits, starting each one of them by the special separator symbol #, and the i-th block being the image of the i-th word of Γ^n. In the special case in which $m = 1$, i.e. when f is a boolean function, it is not necessary to use the separators and the i-th bit will represent the image of the i-th word of Γ^n. In the case when $m > 1$ it is not hard to see that the problem of whether a given input is a correct encoding of a function is in $\forall^l LT$.

We will interpret the blocks in the encoding of the function as integers sometimes. For instance, the "non-zero string" problem of example 9 can be reworded in this setting as "not identically zero function" from Γ^n to Γ for some n. Given a (finite or infinite) function f and a finite set A, we denote $f|_A$ the restriction of f to A; this set A will usually be of the form Γ^n.

The next theorems give sufficient conditions for a problem on finite functions to be hard for $NLT = \exists^l LT$ and $\mathbf{C}^l LT$. These conditions will be given by the existence of "$\exists$-critical" (resp. "C-critical") functions. We need some definitions.

Let Q be a property on finite functions, and let f and g be two linear-time computable functions, $f,\ g : \Gamma^\star \mapsto \Gamma^\star$, such that for every n and for every $z \in \Gamma^n$, $|g(z)| = |f(z)| = m$ where m is a power of 2 that only depends on n. We will say that the pair (f, g) is $\exists$-critical for Q if for every $n \in \mathbb{N}$ the following conditions hold:

i/ the restriction $f|_{\Gamma^n} \notin Q$;

ii/ for every nonempty set $B \subseteq 0\Gamma^{n-1}$, that is, every nonempty set of words of length n beginning with 0, the function $h_B|_{\Gamma^n} \in Q$, where

$$h_B(z) = \begin{cases} f(z) & \text{if } z \notin B \\ g(z) & \text{if } z \in B \end{cases}$$

Most applications will take $m = 1$. We will identify each property Q with the decisional problem of deciding whether the input has the property Q. Now the following powerful theorem yields easily many complete problems for $\exists^l LT$.

11. *Theorem.* Let Q be a property of finite functions. If there exists a $\exists$-critical pair (f, g) of functions for Q, then Q is $\exists^l LT$-hard for the LT-reducibility.

Proof. We show how to reduce L_1 to Q. Given x, we must indicate a value y such that $x \in L_1 \iff y \in Q$, and explain how to compute in logarithmic time the function φ that finds the bits of y.

We use the following notation. Let n be the minimum integer such that $|x| \leq 2^{n-1}$, so that $n < \log|x| + 2$; it can be computed as in example 1. Let the length of the values of f and g on inputs of length n be m, a power of 2. Observe that divisions by m are mere shifts, and thus can be easily computed. The value m itself is computable by evaluating f on 0^n, in time linear in n and thus logarithmic on $|x|$.

The value y is to be interpreted as a finite function from Γ^n to Γ^m, interleaving values of f and values of g. Its length will be $|y| = m2^n$: a total of 2^n blocks of m bits, each corresponding to a value of f or g.

For the first $|x|$ blocks (less than half the blocks), the corresponding bit of x determines whether the block comes from f or from g: if the k-th bit of x is $x_k = 1$, then y will have a value of g as k-th block; otherwise it will have a value of f. The remaining blocks are extra values of f, added in order to complete y.

Thus the i-th bit of y is computed by finding out the block r on which it falls, and its relative position j within the block: those are quotient and remainder of i divided by m; then, checking x_r to see if f or g is being used for that block (for the first $|x|$ blocks only), and extracting the j-th bit of the value of f or g corresponding to the block.

The complete description of φ is as follows. Bits of words (and also blocks) are assumed to be numbered starting from zero to simplify the arithmetics.

$$\varphi(x,i) = \begin{cases} g(r)_j & \text{if } r = \lfloor \frac{i}{m} \rfloor \text{ and } j = i \bmod m \\ & \text{and } i < m|x| \text{ and } x_r = 1 \\ f(r)_j & \text{if } r = \lfloor \frac{i}{m} \rfloor \text{ and } j = i \bmod m \\ & \text{and } i < m|x| \text{ and } x_r = 0 \\ f(r)_j & \text{if } r = \lfloor \frac{i}{m} \rfloor \text{ and } j = i \bmod m \\ & \text{and } m|x| \le i < m2^{n-1} \\ f(r)_j & \text{if } r = \lfloor \frac{i}{m} \rfloor \text{ and } j = i \bmod m \\ & \text{and } m2^{n-1} \le i < m2^n \end{cases}$$

Here $f(r)$ is interpreted as writing r in binary, padding with leading zeros up to length n if necessary, and applying f, and similarly for $g(r)$. Computing either f or g takes time linear on $|r| < \log|x| + 2$. Observe that the last line covers exactly those blocks for which r begins with a 1, i.e. half of them.

Let us see that this is a reduction. If $x \in L_1$ then there is at least a 1 in some position of x, and then the list of the function values will have one value of g for an argument beginning with 0. By condition ii/ in the definition of criticality, it follows that the function represented by y belongs to Q. Conversely, if $x \notin L_1$, then y will be exactly the encoding of $f|_{\Gamma^n}$, and by condition i/ in the definition of criticality, y is not in Q. ■

12. *Remark.* The fact that the value m is a power of 2 is not essential, since the fact that the divisions are shifts may be guaranteed in other ways. For instance, if for all z the critical pair fulfills $|f(z)| = |g(z)| = |z|$ instead, we would get $m = n$; in the reduction we let n be the minimum integer such that $|x| \le 2^{n-1}$ and n is a power of 2. It can be computed in the following way: first, as in example 1, find the minimum integer k such that $|x| \le 2^k$; then n is the string $10^{|k|}$ of a single one followed by $|k|$ zeros. The bound on $n = |r|$ would be $2\log|x| + 2$. The proof goes through.

13. *Remark.* Similarly, the fact that the ratio of actual bits affected by the reduction is one half, as implied by $B \subseteq 0\Gamma^{n-1}$, is not crucial either. The reader can check that we could have taken, for instance, $B \subseteq 0^{3n/4}\Gamma^{n/4}$. Of course $|x|$ must be less than the size of the part of the output indexed by B; in this case, for instance, taking n so that $2^{n/8} \le |x| < 2^{n/4}$ works. The bound on $|r|$ is still linear on $\log|x|$.

We will use theorem 11 for classifying the next problems. Observe that in order to show that a problem is $\exists^l LT$-hard, we only need to give a pair of $\exists$-critical functions (f,g) for the problem. Similarly, it can be proved that a problem is $\forall^l LT$-hard by showing that its complement is $\exists^l LT$-hard. The following four problems are $\forall^l LT$-complete: we give an $\exists$-critical pair for the complement. It is left to the reader to prove that these problems are in $\forall^l LT$ and that the pairs are actually $\exists$-critical for their complements.

14. *Example.* Constant function.

$L_3 = \{\xi : \Gamma^n \mapsto \Gamma^m$ such that ξ is a constant function $\}$

$\exists$-critical functions for the complement: $f(z) = 0$, $g(z) = 1$.

15. *Example.* Injectiveness.

$L_4 = \{\xi : \Gamma^n \mapsto \Gamma^n$ such that ξ is an injective function $\}$

$\exists$-critical functions for the complement: $f(z) = z$, $g(z) = 1^{|z|}$. Here we use the condition in the definition of $\exists$-critical that words of B begin with 0 and remark 12.

16. *Example.* Increasing function.

$L_5 = \{\xi : \Gamma^n \mapsto \Gamma^n$ such that ξ is strictly increasing $\}$

$\exists$-critical functions for the complement: $f(z) = z$, $g(z) = 1^{|z|}$.

17. *Example.* Conservation of parity.

$L_6 = \{\xi : \Gamma^n \mapsto \Gamma^m$ such that for every $z \in \Gamma^n$, $lb(z) = lb(\xi(z))\}$

where we denote by $lb(z)$ the last bit of z. $\exists$-critical functions for the complement: $f(z) = lb(z)$, $g(z) = 1 - lb(z)$.

The following examples are taken from [6], where actually only succinct versions are considered. We will consider standard problems here, and obtain the results of [6] in the next section as applications of the Conversion Lemma. It should be observed that the problem "nonemptiness" of [6], under our encoding of undirected graphs, is exactly the "nonzero string" problem L_0. In the next examples the functions ξ are identified with the adjacency matrix of a graph on $n = 2^k$ vertices; each potential edge is thus identified by exactly $2k$ bits, k for each endpoint (padding with leading zeros if necessary).

18. *Example.* Triangle.

$L_7 = \{\xi : \Gamma^{2k} \mapsto \Gamma$ has a triangle $\}$

$\exists$-critical functions:

$$f(z_1z_2) = \begin{cases} 1 & \text{if } z_1 = n/2 = 2^{k-1}, \text{ i.e. } z_1 \text{ is a one and } k-1 \text{ zeros} \\ 0 & \text{otherwise} \end{cases}$$

representing a "star" graph in which vertex 2^{k-1} is connected to all other vertices and no other edge exists, and $g(z_1z_2) = 1$. A value taken from g for an argument beginning with 0 corresponds to adding an edge between two vertices different from (smaller than) 2^{k-1}. Since both are connected to it, a triangle appears. We accept as a triangle here a trivial one formed by an edge and a self-loop.

19. *Example.* k-path.

$L_8 = \{\xi : \Gamma^{2k} \mapsto \Gamma$ has a k-path $\}$

We use remark 13 to assume that $B \subseteq 0^{3k/2}\Gamma^{k/2}$. The function f will represent a graph with $k-1$ consecutive edges, with an endpoint at vertex 0 and all other vertices past $n/2$. As soon as an edge is added from vertex 0 to any vertex strictly between 0 and $n/2$, a k-path appears.

The $\exists$-critical functions are: $f(z_1z_2) = 1$ if $z_1 = 0$ and $z_2 = n/2$, or if z_1 and z_2 are consecutive and strictly between $n/2$ and $n/2 + k - 2$, and 0 otherwise; and $g(z_1z_2) = 1$. Similarly, for the problem k-cycle, it is enough that in the graph defined by f an edge is added connecting 0 to the last vertex in the path.

20. *Example.* Maximum degree $\Delta \geq k$

$L_9 = \{\xi : \Gamma^{2k} \mapsto \Gamma \text{ with } \Delta(\xi) \geq k\}$

We interpret the degree of ξ at $z \in \Gamma^k$ as the number of ones in the column $\{f(z'z) : z' \in \Gamma^k\}$. The maximum degree of ξ, $\Delta(\xi)$, is the maximum of the degrees of ξ at any z. The correspondence with the graph-theoretic notion is apparent.

$\exists$-critical functions: $f(z_1 z_2) = 1$ if and only if $z_1 > n - k + 1$, and $g(z) = 1$.

Now we present some examples taken from [29], concerning problems on finite sets of integers. We represent the sets by their characteristic functions as subsets of Γ^n, where n is large enough to write any integer from the set with exactly n bits (again using leading zeros as padding). In [29], succinct versions were considered, actually in terms of various compact description languages. Again we will consider standard problems here, and leave the results on succinct versions to the next section as applications of the Conversion Lemma. Again the "nonemptiness" problem for sets is L_0. For those problems involving two sets, we encode both in the same function: the restriction of the function to $0\Gamma^{n-1}$ will encode the first and the restriction to $1\Gamma^{n-1}$ the second. Sometimes single elements appear as data in [29]: we fix their value to appropriate constants without loss of generality.

21. *Example.* Critical element.

An element a is critical for a set if a not being in the set implies that it is empty [29].

$L_{10} = \{\xi : \Gamma^n \mapsto \Gamma \text{ s.t. } 2^n \text{ is critical }\}$

$\exists$-critical functions for the complement: $f(z) = 0$ and $g(z) = 1$.

22. *Nonempty intersection.*

$L_{11} = \{\xi : \Gamma^n \mapsto \Gamma \text{ s.t. } \exists z f(0z) = f(1z) = 1\}$

$\exists$-critical functions: $f(0z) = 0$, $f(1z) = 1$; $g(z) = 1$.

23. *Example.* Maximum below a boundary.

Is a the maximum element in the set smaller than b?

$L_{12} = \{\xi : \Gamma^n \mapsto \Gamma$ s.t. 0 is the maximum argument z smaller than or equal to 2^n for which $\xi(z) = 1\}$

$\exists$-critical functions for the complement: $f(0) = 1$, $f(z) = 0$ for $z > 0$, $g(0) = 0$, $g(z) = 1$ for $z > 0$.

24. *Example.* Subset.

$L_{13} = \{\xi : \Gamma^n \mapsto \Gamma \text{ s.t. } \forall z f(0z) \leq f(1z)\}$

$\exists$-critical functions for the complement: $f(z) = 0$ and $g(z) = 1$.

25. *Example.* Equality.

$L_{14} = \{\xi : \Gamma^n \mapsto \Gamma \text{ s.t. } \forall z f(0z) = f(1z)\}$

$\exists$-critical functions for the complement: $f(z) = 0$ and $g(z) = 1$.

26. *Example.* Set connectedness.

The set represented by ξ is connected if $\forall x, y, z$ with $x < y < z$,

$$\xi(x) = 1 \wedge \xi(z) = 1 \Rightarrow \xi(y) = 1.$$

$L_{15} = \{\xi : \Gamma^n \mapsto \Gamma$ is connected $\}$

$\exists$-critical functions for the complement: $f(z) = 1$ if and only if $z_1 > 2^{n-1}$; and $g(z) = 1$.

We move now to a similar sufficient condition for $\mathbf{C}^l LT$ completeness, as announced before. It has a similar structure to that of theorem 11. The analogous condition is as follows.

Let Q be a property on finite functions, and let f and g be two linear-time computable functions, $f, g : \Gamma^\star \mapsto \Gamma^\star$, such that for every $x \in \Gamma^n$, $|g(x)| = |f(x)| = m$ where m is a power of 2 that depends only on n. We will say that the pair (f, g) is $\mathbf{C}$-critical for Q if the following condition holds:

For every nonempty set $B \subseteq \Gamma^n$, the function $h_B|_{\Gamma^n} \in Q$ if and only if $||B|| > 2^{n-1}$, where as before

$$h_B(x) = \begin{cases} f(x) & \text{if } x \notin B \\ g(x) & \text{if } x \in B \end{cases}$$

27. *Theorem.* Let Q be a property of finite functions. If there exists a $\mathbf{C}$-critical pair (f, g) of functions for Q, then Q is $\mathbf{C}^l LT$-hard for the LT-reducibility.

Proof. The proof is analogous to that of theorem 11, reducing instead problem L_2 to Q. This function φ that computes the bits of the image y is quite similar to that of theorem 11. Essentially, for a given string x, here we have to produce a list having as many values of f as 0's in x, and as many values of g as 1's in x. We must be careful also that in the part that pads up y to complete a finite function we put as many values of f as values of g. The resulting word y coding the resulting finite function will have size $m2^n$, where n is the smallest integer such that $|x| \leq 2^n$ and $m = |f(0^n)|$, m a power of 2. Instead of describing the function in detail as we did for theorem 11, we just give the definition:

$$\varphi(x, i) = \begin{cases} g(r)_j & \text{if } r = \lfloor \frac{i}{m} \rfloor \text{ and } (i \bmod m) = j \\ & \text{and } ((i < m|x| \text{ and } x_r = 1) \text{ or } (m|x| \leq i < m2^n \text{ and } r \text{ odd})) \\ & \\ f(r)_j & \text{if } r = \lfloor \frac{i}{m} \rfloor \text{ and } (i \bmod m) = j \\ & \text{and } ((i < m|x| \text{ and } x_r = 0) \text{ or } (m|x| \leq i < m2^n \text{ and } r \text{ even})) \end{cases}$$

For a given x, x has more 1's than 0's if and only if y encodes a function with the values of g arising more frequently than the values of f, which by the condition in the definition of $\mathbf{C}$-critical function implies that $y \in Q$. ∎

28. *Remark.* Again there is no mystery about the lower bound 2^{n-1} on $||B||$. By adjusting the ratio of f's and g's in the section between $m|x|$ and $m2^n$, other thresholds can be used in the same simple manner.

This theorem can be easily applied to a number of problems in the straightforward way. The reader can probably suggest several natural examples along the lines of the previously presented ones. In particular, interesting examples are the cardinality problem and the number of components problem, defined in [29], where it is proved that in succinct form they are complete for $\mathbf{C}^p P$. Here we infer easily from theorem 27 that their standard forms are complete for $\mathbf{C}^l LT$.

In the next section we will discuss what happens to the succinct versions of all the problems studied here: all of them will increase in complexity by exactly one exponential, and this will follow from the Conversion Lemma.

6. COMPLETE SETS AT HIGHER LEVELS

We give now some more examples of function problems that are complete in some other levels of LCH, sketching the proof of their completeness. We treat some classes at what could be called the second level of the hierarchy, and then discuss complete problems in any of the classes of some subhierarchies of LCH. Relaxing certain bounds, we will obtain P-complete problems. Finally, we apply the Conversion Lemma to translate complete sets for LCH into succinct versions that turn out to be complete for CH.

The Second Level

We understand by "the second level" those classes that can be defined using two quantifiers. We give some problems whose complexity corresponds to this level.

29. *Example.* Surjective function.

$L_{16} = \{\xi : \Gamma^{2n} \mapsto \Gamma^{2n}$ such that ξ is surjective on $0^n\Gamma^n\}$

First we prove that $L_{16} \in \forall^l\exists^l LT$. For a given $w \in \Gamma^\star$, $|w| = 2n{\cdot}2^{2n}$, it has to be checked that every value of the form $0^n w$ appears in the given string and starting in a correct position. This can be expressed using quantifiers, in the following way:

$$\begin{aligned} w \in L_{16} \iff & w \text{ is correct encoding of a function } \xi : \Gamma^{2n} \mapsto \Gamma^{2n} \text{ and} \\ & \forall y(|y| = n) : \exists i : w_i = \#, \text{ and } w_{i+1}\cdots w_{i+2n} = 0^n y \end{aligned}$$

Let us see that L_{16} is hard for $\forall^l\exists^l LT$. Let $L \in \forall^l\exists^l LT$. There is a constant c and a LT-predicate P such that for every $x \in \Gamma^\star$,

$$x \in L \iff \forall^{lc} y : \exists^{lc} z : P(x, y, z)$$

We give a reduction from L to L_{16}. For a given x, let n be the smallest power of 2 greater than or equal to $c\lceil\log|x|\rceil$. The image $g(x)$ will have size $2n{\cdot}2^{2n}$. The idea is that in this image, for every y, we will make the substring $0^n y$ to appear if there is a z such that $P(x, y, z)$, constructing the substring 1^{2n} otherwise. Giving the function φ bit by bit as we did in the previous examples makes it difficult to understand it;

therefore we give it in "pieces" of length $2n$. Recall from the preliminaries that we denote by $[w]$ the integer whose binary expansion, padded up to length $2n$, is w.

$$\varphi(x,[yz]\cdot 2n+1)\cdots\varphi(x,([yz]+1)\cdot 2n) = \begin{cases} 0^n y & \text{if } |y|=|z|=n \text{ and } P(x,y,z) \\ 1^{2n} & \text{if } |y|=|z|=n \text{ and } \neg P(x,y,z) \end{cases}$$

Observe that again we need the fact that n is a power of 2, since by giving function φ in pieces of length $2n$, we are actually hiding a division by $2n$. The transformation from an integer i into the corresponding string is trivial since one just has to add 0's to the left until the string has length $2n$.

30. *Example.* Minimum degree $\delta \geq k$

$L_{17} = \{\xi : \Gamma^{2k} \mapsto \Gamma \text{ with } \delta(\xi) \geq k\}$

The degree of ξ at $z \in \Gamma^k$ was defined in the example 20. The minimum degree of ξ, $\delta(\xi)$, is the minimum of the degrees of ξ at any z. Again there is a correspondence with the graph-theoretic notion. This problem is complete for Π_2-alternating logarithmic time. We omit the proof: it is similar to the previous one.

31. *Example.* Large injective restriction.

$L_{18} = \{\xi : \Gamma^{2n} \mapsto \Gamma^{2n} \text{ s.t. there is an injective restriction of } \xi \text{ of size} \geq 2^n\}$

L_{18} is complete in $\mathbf{C}^l\exists^l LT$. The proof follows again guidelines similar to the previous ones.

Complete Problems for Σ^l_k: a Tiling Game

We will consider a finite set of strings $T \subseteq \Gamma^{2s}$ as the set of tiles of a one-dimensional domino game. Given a string $x \in T$, a second string $y \in T$ will match with x if the last s digits of x coincide with the first s digits of y. We will also consider a number $m \in \mathbb{N}$ and two players, Constructeur and Saboteur as in [4], building a tiled row of dominoes. Given a starting tile, alternately, each of the two players selects a domino that matches with the right part of the last selected tile, and adds it to the row. The aim of Constructeur is to build a tiled row of $m+1$ tiles, while the aim of Saboteur is to prevent Constructeur from reaching his aim. We investigate different versions of this tiling game. Similar tiling games have been independently defined by Grädel in [9], where they are used as tool to study the complexity of boolean algebras. For each odd m we define the following problem:

$R(m)$: *Tiling game of length* m ($m > 1$ and odd). Consider a tiling game of 2^{mn} tiles of length $2mn$ given one after another in a string and separated by the marker #. The starting tile is given by the first $2mn$ digits of the string. If Constructeur starts the game, can he complete a tiled row of $m+1$ tiles?

32. *Proposition.* $R(m)$ is Σ^l_m-complete.

Proof. We prove first that $R(m) \in \Sigma^l_m$. Given $w \in \Gamma^*$, $|w| = 2mn\cdot 2^{mn}$, we can write the decision problem as a string of m quantifiers followed by a log-time predicate, so that the meaning of the expression is that Constructeur has a way to select tiles in which Saboteur either plays unfair (placing tiles that do not match) or cannot prevent Constructeur from making the tiled row:

$$
\begin{aligned}
w \in R(m) \iff\ & w \text{ has the correct encoding and } \exists i_1 \forall i_2 \ldots \exists i_m : \\
& w_{i_1} = w_{i_2} = \ldots = w_{i_m} = \# \text{ and} \\
& w_{i_1+1} \ldots w_{i_1+2mn} w_{i_2+1} \ldots w_{i_2+2mn} \ldots w_{i_m+1} \ldots w_{i_m+2mn} \\
& \text{is a correct row of the domino game, or for some even } j \\
& w_{i_j+1} \ldots w_{i_j+2mn} \text{ does not match with } w_{i_{j-1}+1} \ldots w_{i_{j-1}+2mn}.
\end{aligned}
$$

To prove that $R(m)$ is hard for Σ_m^l (m odd), let L be a language in Σ_m^l. For a certain constant c and a log-time predicate P, and for every $x \in \Gamma^\star$,

$$x \in L \iff \exists^{lc} u_1 \forall^{lc} u_2 \ldots \exists^{lc} u_m : P(x, u_1, \ldots, u_n)$$

Let n be the smallest power of two greater than or equal to $c\lceil \log |x| \rceil$; for any $k < 2^n$, represent by $(k)_n$ the binary expansion of length n of k. We will reduce L to $R(m)$. For this we will construct a domino game of $2^{2(m+1)n}$ tiles of size $2(m+1)n$ in such a way that in any row, the first m tiles can always be placed. The tile $m+1$, corresponding to Constructeur, is related to the predicate P and to the tiles placed before, and if both players have played right, it can only be placed if $x \in L$. This will be argued after the definition of the function φ. Again, for clarification, we will describe the function in pieces, each piece corresponding to a tile of the domino game in the reduction. Again $[\cdot]$ represents the standard mapping from strings in $\Gamma^{2(m+1)n}$ into integers.

$$\varphi(x, [u_1 u_2 \ldots u_m u_{m+1}] \cdot 2(m+1)n + 1) \ldots \varphi(x, ([u_1 u_2 \ldots u_m u_{m+1}] + 1) \cdot 2(m+1)n) =$$

$$
= \begin{cases}
\overline{\underline{|(m)_n 0^n 0^n \ldots 0^n|(0)_n 0^n 0^n \ldots 0^n|}} & \text{if } u_j = (0)_n \text{ for } j \in \{1 \ldots m+1\} \\
\overline{\underline{|(0)_n 0^n 0^n \ldots 0^n|(1)_n u_1 0^n \ldots 0^n|}} & \text{if } u_j = (1)_n \text{ for } j \in \{2 \ldots m+1\} \\
\overline{\underline{|(1)_n u_1 0^n \ldots 0^n|(2)_n u_1 u_2 0^n \ldots 0^n|}} & \text{if } u_j = (2)_n \text{ for } j \in \{3 \ldots m+1\} \\
\overline{\underline{|(2)_n u_1 u_2 0^n \ldots 0^n|(3)_n u_1 u_2 u_3 0^n \ldots 0^n|}} & \text{if } u_j = (3)_n \text{ for } j \in \{4 \ldots m+1\} \\
\ldots & \\
\ldots & \\
\overline{\underline{|(m-2)_n u_1 u_2 \ldots u_{m-2} 0^n 0^n|(m-1)_n u_1 \ldots u_{m-2} u_{m-1} 0^n|}} & \\
 & \text{if } u_j = (m-1)_n \text{ for } j \in \{m, m+1\} \\
\overline{\underline{|(m-1)_n u_1 u_2 \ldots u_{m-2} u_{m-1} 0^n|(m)_n u_m 0^n \ldots 0^n|}} & \\
 & \text{if } u_{m+1} = (m)_n \text{ and } P(x, u_1, u_2 \ldots u_m) \\
\overline{\underline{|(m)_n 0^n \ldots 0^n|(0)_n 0^n \ldots 0^n|}} & \text{otherwise}
\end{cases}
$$

We will see that Constructeur can build a tiled row of $m+1$ tiles if and only if $x \in L$. Observe that the only tiles that depend in the predicate P have the form $\overline{\underline{|(m-1)_n u_1 \ldots u_{m-1} 0^n|(m)_n u_m 0^n \ldots 0^n|}}$. In these tiles is encoded the whole history of the game, i.e., the $u's$ will correspond to the quantified variables is the definition of L. The remaining tiles will always be in our set of dominoes; this means that the tiled row constructed in the game will always have length at least m. The string of length n written at the left of every tile guarantees that the j-th tile in the row encodes an election u_{j-1} for the quantifier $j-1$. It will be possible to place the tile $m+1$ if and only if there is a choice of u_m such that $P(x, u_1, \ldots u_m)$. If $x \in L$ then $\exists u_1 : \forall u_2 \ldots \exists u_m : P(x, u_1, \ldots u_m)$. Considering Saboteur's selections, Constructeur only needs to select the tiles that codify the values corresponding to the existential quantifiers in the formula. Conversely, if $\forall u_1 : \exists u_2 \ldots \forall u_m : \neg P(x, u_1 \ldots u_m)$, then Saboteur can select tiles so that the m-th domino in the row is $\overline{\underline{|(m-2)_n u_1 \ldots u_{m-2} 0^n 0^n | (m-1)_n u_1 \ldots u_{m-2} u_{m-1} 0^n|}}$ and for any u_m the tile $\overline{\underline{|(m-1)_n u_1 \ldots u_{m-2} u_{m-1} 0^n | (m)_n u_m 0^n \ldots 0^n|}}$ is not in our set of dominoes. It follows that Constructeur cannot finish the row. ■

We require the game to have an odd number of moves since we want Constructeur to start and finish the game. This is why the related problem lies in an odd level of the logarithmic time hierarchy. Analogous games could be defined for the even levels.

Complete Problems for $\mathbf{C}_k^l$: the Green Tree

Consider a tree whose leaves can be labeled with either a 1 (the leaf is green) or a 0 (the leaf is brown). We will say that an interior node of the tree is green if more than half of its direct descendants are green. A tree is green if its root is green. We will restrict ourselves to complete trees having degree 2^n, $n \in \mathbb{N}$ i.e. each interior node has 2^n direct descendants. A complete tree with its leaves labeled, having k levels and with every node having 2^n direct descendants will be represented by a list of 2^{kn} bits, each representing the label of one of the leaves. The problem is as follows:

$T(k)$: *Green tree of height k.* Given a list of 2^{kn} bits, does it encode a green tree of k levels?

33. *Proposition.* $T(k)$ is $\mathbf{C}_k^l$ complete.

Proof. We prove that $T(k) \in \mathbf{C}_k^l$. Given $w \in \Gamma^\star$, the length of w can be checked using standard techniques. Suppose $|w| = 2^{kn}$, $n \in \mathbb{N}$.

$$w \in T(k) \iff \mathbf{C} i_1(|i_1| = n) : \mathbf{C} i_2(|i_2| = n) \cdots \mathbf{C} i_k(|i_k| = n) : w_{(i_1 i_2 \ldots i_k)} = 1$$

To see that $T(k)$ is hard for $\mathbf{C}_k^l$, let $L \in \mathbf{C}_k^l$. There is a constant c and a polynomial-time predicate P such that for every $x \in \Gamma^\star$,

$$x \in L \iff \mathbf{C}^{lc} u_1 : \mathbf{C}^{lc} u_2 \cdots \mathbf{C}^{lc} u_k : P(u_1 \ldots u_k)$$

where without loss of generality we assume all thresholds to be 1/2. We give the reduction from L to $T(k)$. Let n be the smallest power of 2 greater than or equal to $c\lceil \log |x| \rceil$. The image $g(x)$ will have size 2^{kn}, and the function φ defining its bits is simply

$$\varphi(x, (u_1 u_2 \ldots u_k)) = \begin{cases} 1 & \text{if } P(u_1 u_2 \ldots u_k) \\ 0 & \text{if } \neg P(u_1 u_2 \ldots u_k) \end{cases}$$

Unbounded Versions of the Tiling Game and the Green Tree

This subsection is an aside with little relation with the rest of the paper. Observe that the last two problems studied are "bounded" in the sense that we only allow a tiling game to have a constant number of moves, or a tree to have a constant depth. What happens if we do not enforce these limitations? We show that in this case the problems are complete for P.

R': Given a set of tiles of length k, a starting tile t_1 and an ending tile t_2, can Constructeur finish a tiled row from t_1 to t_2, with Saboteur trying to prevent him from reaching his aim, and moving alternatively?

$T' = \{k\#w : k \in \mathbb{N}, w \in \Gamma^{2^{kn}} \text{ s.t. } w \in T(k)\}$

34. *Proposition.* R' and T' are LT-complete for P.

Proof. We see the proof for T'; the one for R' is easier.

To see that $T' \in P$, given $k\#w$, first check $|w| = 2^{kn}$ for a certain $n \in \mathbb{N}$. Then write a string of size $2^{k(n-1)}$ with a labeled list of the nodes at distance 1 from the leaves (encoding them by a 1 if they are green or by a 0 if they are brown), and iterate k times the process until the root is reached. Since $|w| = 2^{kn}$ the process is polynomial with respect to $|w|$.

To see that T' is hard for P, we reduce to T' the problem G described below. G is a simplified version of the problem "Game" which is shown to be complete in P in [12]. The completeness of G for the class P can be proved using the characterization of P in terms of alternating log-space machines [2].

G: Given a two-player game, encoded by a complete tree with every node having exactly n direct descendants and with its leaves labeled 0 (losing position for the first player) or 1 (winning position for the first player), the two players select alternatively a path going from the standing node to one of its descendants. Does the first player have a winning strategy that takes him from the root to a leaf labeled 1?

To reduce G to T', we use a trick similar to the one used by Gill [8] to show that $NP \subseteq PP$. Given a complete tree t with k levels and with each node having exactly n direct descendants (t can be encoded with 2^{kn} bits), we can construct a new tree t', with every node having $2n$ direct descendants. For l equals 1 to k, connect every node of level l in t, with n new trees of $l-1$ levels, being all the new trees green if l corresponds to a level in which the first player moves, and all ot them brown if l corresponds to a move of the second player. The resulting tree t', that can be encoded with 2^{2kn} bits, is green if an only if in t the first player has a winning strategy. ■

Succinct Representations

We have fruitfully discussed completeness for many classes of LCH. Let us consider now succinct versions of these problems. As before, the succinct version sA of a problem A on finite functions gets as input a boolean circuit computing a function from Γ^n to Γ^m, and consists of solving the standard problem for the finite function computed by the circuit.

Let us first prove a simple extension of lemma 2 to the classes defined by means of alternating quantifiers.

35. *Lemma.* If L is a problem in K, $K \in LCH$, then the succinct version of L, sL, is in K', being K' the corresponding exponentially larger class in CH.

Proof. Let $K \in LCH$ and $L \in K$. By definition, for a certain $k \in \mathbb{N}$, a constant c_1, and a log-time predicate P, and for any $x \in \Gamma^*$,

$$x \in L \iff Q_1^{lc_1} i_1 : Q_2^{lc_1} i_2 \cdots Q_k^{lc_1} i_k : P(x, i_1, i_2 \ldots i_k)$$

where each Q_i is an existential, universal, or counting quantifier. Let sL the succinct version of L, and C_x the boolean circuit representation of an instance x. Supose that C_x receives n bits as input, and produces m bits as output. C_x encodes then a string x of $m2^n$ bits, and since $m + n \leq |C_x|$, we have $2^{|C_x|} \geq |x|$. It follows that for some polynomial p

$$C_x \in sL \iff Q_1^p i_1 : Q_2^p i_2 \cdots Q_k^p i_k : P'(x, i_1, i_2 \ldots i_k)$$

being P' a polynomial-time predicate (respect to C_x) working like P, except that when P queries a bit in a position i of x, P' queries the bit indirectly from C_x using the following process: compute $r = \lfloor \frac{i}{m} \rfloor$, the input that produces the part of x that contains its i-th bit; then feed into the circuit the r-th string of length n in lexicographical order, evaluate it, and select from the output the bit in position $i \bmod m$. This is the i-th bit of x. This simulation process can be done in polynomial time with respect to C_x. It follows that $sL \in K'$. ■

As a consequence of lemma 35 and theorem 5 (essentially, the Conversion Lemma), the succinct versions of the problems that we have shown to be complete in different classes of LCH are complete in the corresponding classes of CH with respect to the m-reducibility. This includes all the NP-completeness and co-NP-completeness results of [6] and [29] and most of their other results for other classes. The remaining ones are settled in the next section.

36. *Corollary.* The succinct versions of the problems $L_1 \ldots L_{18}$, $R(m)$, and $T(k)$ are complete in the corresponding classes of CH with respect to the m-reducibility.

7. UNDIRECTED ACCESSIBILITY AND RELATED PROBLEMS

The main result in this section is a construction of a graph associated to a given quantified boolean formula, which has the following property: for two selected nodes in the graph, there is a path joining them if and only if the quantified boolean formula is true. Although the size of the graph is exponential in the formula, we will see that it can be described by a considerably shorter boolean circuit which can be constructed from the formula in polynomial time.

We employ this construction to present an ad-hoc proof of the $PSPACE$-hardness of the succinct version of undirected accessibility. From it, combining the Conversion Lemma with known LT-reducibilities among graph problems, we will obtain further classifications of succinct versions of graph problems, including planarity, bipartiteness, connectivity, acyclicity, Eulerian path, and perfect matchings in bipartite graphs. We end the section by discussing the interesting particular case of bounded degree planarity.

The result regarding undirected accessibility can be inferred from the work of [16] on symmetric computation, since they prove that this problem is complete for the class SL of symmetric logarithmic space, and that this class includes L. Therefore, by the Conversion Lemma, the succinct version is $PSPACE$-complete. The interest of the proof we present here is: first, that it is very intuitive and clear, independent of the complex simulations of [16]; second, that it is essentially a natural (albeit nontrivial) extension of a technique of [6] which the authors did not expect to reach any further, as mentioned in their conclusions; third, that it can be immediately seen to apply to planarity for bounded degree graphs, a case for which we are able to completely settle the problem.

Before presenting the construction, let us define *quantified boolean formulas.* A boolean formula is either a boolean variable, its negation, a boolean constant "true" or "false," the conjunction of two boolean formulas, or the disjunction of two boolean formulas. An assignment of boolean values to the variables *satisfies* the formula if and only if once the substitution is made, the formula evaluates to "true." Quantified boolean formulas are boolean formulas preceded by a string of quantifiers (prenex form), each refering to one of the variables of the formula, so that no free variables remain. The allowed quantifiers are $\forall$ and $\exists$. Such a formula is evaluated as follows: $\forall v\, F$ is true if, when substituting "true" and "false" for v, both resulting formulas are true; and $\exists v\, F$ is true if, when substituting "true" and "false" for v, at least one of the resulting formulas is true. Otherwise, the formula evaluates to "false."

The problem QBF is the following: given a quantified boolean formula, decide whether it evaluates to "true." More information about quantified boolean formulas and the problem QBF can be found in [1], including the following fact.

37. *Theorem.* QBF is *PSPACE*-complete.

We present now the construction announced above. We describe it inductively on the form of the formula, by induction on the number of quantifiers. The nodes are identified by sequences of binary numbers. Without loss of generality, we assume that the variables are numbered according to their position in the quantifier string, the innermost being smallest.

Induction Basis. The formula has no quantifiers. Since no free variables are allowed, it evaluates either to "true" or to "false." The associated graph consists of two vertices, labeled 0 and 1, joined by an edge in case the fromula is true, or disconnected if the formula is false. Arbitrarily declare *source node* the node 0 and *target node* the node 1.

Induction Step. We have two cases depending on the outermost quantifier. Let F be the given formula, and assume first that F is $\exists v_i\, F'$. Construct F_0 and F_1 substituting "false" and "true," respectively, for v_i, and inductively consider the graphs associated to them, G_0 and G_1. Re-label all nodes in these graphs by concatenating "$i00$;" and "$i10$;" respectively to their labels. Create two more vertices with labels $i01$ and $i11$. Declare the first *source node* and join it to the source nodes of G_0 and G_1. Declare the other *target node* and join it to the target nodes of G_0 and G_1. The case of F being $\forall v_i\, F'$ is similar, but the graphs are connected in series instead of in parallel, i.e. the new source node is linked to the source of G_0, the target of G_0 is linked to the source of G_1, and the target of G_1 is linked to the new target node.

It is easy to see by induction that the source and target nodes of the obtained graph are connected if and only if the formula evaluates to true. The graph requires exponential time to be constructed, and is itself of size exponential in the size of the formula. However, given the formula, it is possible to decide whether an edge exists between two nodes in time polynomial on the labels of the nodes. Assume that the formula has n variables. If both labels consist of n numbers then the edge corresponds to a nonquantified formula, and the labels provide the value of each variable, so that it only remains to evaluate the formula. This can be done in polynomial time [19].

Else, only sources and targets of the same stage or two consecutive stages can be linked, and this is easy to check by looking at one of the quantifiers in the prefix of the formula.

This algorithm for checking whether an edge exists can be presented by a boolean circuit depending only on the formula F, and the circuit can be constructed in polynomial time from F using the techniques of [15] (see also [1], section 5.4). Thus, we have a circuit C_F which can be constructed in time polynomial in the size of F, which represents a graph in which a source and a target node are selected, such that F evaluates to true if and only if there is a path from the source node to the target node in the graph. This is a polynomial time m-reducibility from QBF to the succinct version of undirected graph accessibility, and therefore we obtain the following theorem:

38. *Theorem.* The succinct version of the graph accessibility problem for undirected graphs is *PSPACE*-hard for the m-reducibility.

A similar reduction works for planarity testing:

39. *Theorem.* The succinct version of the planarity problem for undirected graphs is *PSPACE*-hard for the m-reducibility, even when restricted to graphs of bounded degree.

Proof. Given F, construct a circuit describing a complete graph with five nodes, and then substitute the previously constructed graph for one of the edges, identifying the endpoints with the selected source and target nodes. The resulting graph is nonplanar if and only if there is a path between the source and target nodes, if and only if the formula evaluates to true. Observe that the degree of the graph is bounded. ■

Some more results will be derived from known relationships between graph problems. In [3] many problems are compared according to projection and constant depth reducibility. Although their projections are reducibilities different from the ones employed here, it is easily seen that all the reductions that we need are actually LT-reductions. We will use also a reduction from [13], where it is stated for logarithmic space reducibility; from the proof it is easily seen that it is also a LT-reduction.

40. *Theorem.* [13] The undirected graph accessibility problem is LT-reducible to the problem of deciding whether an undirected graph is bipartite.

41. *Theorem.* [3] The undirected graph accessibility problem is LT-reducible to the following problems on undirected graphs: connectivity, having a cycle, having an Eulerian path, and having a perfect matching (the last, for bipartite graphs).

In fact, that reference shows that all these problems except perfect matching are equivalent under appropriate reducibilities. From these two results and the Conversion Lemma, for the hardness part, and lemma 2 for the membership part, and using the facts that acyclicity is in L [10], and connectivity, Eulerian path [3], and bipartiteness [13] are in *NL*, we obtain:

42. *Theorem.* The succinct versions of acyclicity, connectivity, bipartiteness, and Eulerian path are *PSPACE*-complete.

For perfect matching we are left with:

43. *Theorem.* The succinct version of perfect matching is *PSPACE*-hard.

Since perfect matching and planarity are in *P*, their succinct versions are in *EXPTIME*, and therefore are the only ones left in which the upper and lower bounds are still far apart: we only know their *PSPACE*-hardness. However, we can show that a combination of known results proves that planarity of succinctly represented graphs of bounded degree is in *PSPACE*, and therefore *PSPACE*-complete by theorem 39.

44. *Theorem.* Testing planarity of bounded degree graphs is in *NL*, and therefore its succinct version is *PSPACE*-complete.

Proof. In [21] it is shown that this problem is in the third level of the symmetric complementation logarithmic space hierarchy, which is included by definition in the logarithmic space hierarchy. However, Immerman [11] and Szelepcsényi [25] have shown that *NL* is closed under complementation, and therefore the logarithmic space hierarchy coincides with *NL*. Thus, planarity of bounded degree graphs is in *NL*. ■

8. DISCUSSION

We have considered decisional problems in which the input is represented in a nonstandard form. All combinatorial objects can be naturally encoded as binary words, the adjacency matrix of directed graphs being a prime example. Since such encodings can be viewed as boolean functions, they can be represented by a possibly small boolean circuit. In case the objects are particularly regular, such a representation is shorter than the full standard encoding. Following the research initiated in [6], we have studied how this change in the presentation of the input affects the complexity of the decisional problems.

It was known that the succinctly represented problem is at most exponentially harder than the problem on standard encodings. In [6] it was shown that quite simple succinct graph problems are already *NP*-hard; in [20] it was shown that *NP*-complete standard problems (for projection reducibility) become *NEXPTIME*-complete for succinct inputs, and such an exponential blow-up for problems in *P* was stated.

By introducing logarithmic-time reducibility and two counting hierarchies corresponding to logarithmic time and polynomial time, we have been able to find a generalization that carries over to other, computationally much weaker classes and yields in a uniform manner all the known lower bounds and many others.

More precisely, our Conversion Lemma shows that LT-reducibilities among the standard problems translate into m-reducibilities among succinctly represented problems, independently of any membership or completeness property of the sets in any particular complexity class. Together with known upper bounds, we get precise classifications of problems in *LCH* and *CH*.

As can be expected, research relating two subjects sheds light on both. In this case, not only we have now a better understanding of the phenomenon of succinct descriptions via boolean circuits, but also we have gained knowledge about the logarithmic-time counting hierarchy, providing very natural complete problems and powerful sufficient conditions to find new ones, and giving precise classifications for problems about finite functions in *LCH* and in *CH* for their succinct version. The logarithmic-time counting hierarchy deserves deeper study, since the union of its classes is precisely the class TC_0 of problems solved by constant depth threshold circuits, which characterizes certain neural computation patterns and other models of fast parallel computation; moreover, this hierarchy is the only one where actual separation results are known between infinitely many classes.

Natural continuations of this work would consist of the study of other languages for the representation of succinct instances, such as those introduced in [28], [29], and [14], where also comparisons between the descriptive power of those languages are made. Is it possible to characterize the complexity jump derived from the use of these languages in such an exact way as for the boolean circuit model?

9. REFERENCES

[1] J.L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science, vol. 11, Springer-Verlag (1988).

[2] A. Chandra, D. Kozen, L. Stockmeyer: "Alternation". *Journal ACM* **28** (1981), 114–133.

[3] A. Chandra, L. Stockmeyer, U. Vishkin: "Constant depth reducibility". *SIAM Journal on Computing* **13** (1984), 423–439.

[4] B.S. Chlebus: "Domino-Tiling games". *Journal of Computer and System Sciences* **32** (1986), 374–392.

[5] M. Furst, J.B. Saxe, M. Sipser: "Parity, circuits, and the polynomial-time hierarchy". *Mathematical Systems Theory* **17** (1984), 13–27.

[6] H. Galperin, A. Wigderson: "Succinct representations of graphs". *Information and Control* **56** (1983), 183–198.

[7] M. Garey, D. Johnson: *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman (1978).

[8] J. Gill: "Computational complexity of probabilistic Turing machines". *SIAM Journal on Computing* **6** (1977), 675–695.

[9] E. Grädel: "Domino games with an application to the complexity of boolean algebras with bounded quantifier alternation". Proc. 5th Symp. Theor. Aspects of Comp. Sci. LNCS 294, Springer-Verlag (1988), 98–107.

[10] J. Hong: "On some deterministic space complexity problems". *SIAM Journal on Computing* **11** (1982), 591–601.

[11] N. Immerman: "Nondeterministic space is closed under complementation". *SIAM Journal on Computing* **17** (1988), 935–938.

[12] N. Jones, W. Laaser: "Complete problems for deterministic polynomial time". *Theoretical Computer Science* **3** (1977), 105–117.

[13] N.D. Jones, E. Lien, W.T. Laaser: "New problems complete for nondeterministic log space". *Mathematical Systems Theory* **10** (1976), 1–17.

[14] M. Kowaluk, K. Wagner: "Vector language: simple description of hard instances". Proc. Math. Found. of Comp. Sci. LNCS 452, Springer-Verlag (1990), 378–384.

[15] R.E. Ladner: "The circuit value problem is log space complete for *P*". *SIGACT News* **7** (1975), 18–20.

[16] H.R. Lewis, Ch. Papadimitriou: "Symmetric space-bounded computation". *Theoretical Computer Science* **19** (1982), 161–187.

[17] A. Lozano: "*NP*-hardness on succinct representations of graphs". *Bulletin of the EATCS* **35** (1988), 158–163.

[18] A. Lozano, J.L. Balcázar: "The complexity of graph problems for succinctly represented graphs". Proc. Graph-Theoretic Concepts in Comp. Sci. LNCS 411, Springer-Verlag (1989), 277–285.

[19] N. Lynch: "Logspace recognition and translation of parenthesis languages". *Journal ACM* **24** (1977), 583–590.

[20] C.H. Papadimitriou, M. Yannakakis: "A note on succinct representations of graphs". *Information and Control* **71** (1986), 181–185.

[21] J.H. Reif: "Symmetric complementation". *Journal ACM* **31** (1984), 401–421.

[22] M. Sipser: "Borel sets and circuit complexity". Proc. 15th Symp. Theory of Comp. (1983), 61–69.

[23] S. Skyum, L.G. Valiant: "A complexity theory based on boolean algebra". *Journal ACM* **32** (1985), 484–502.

[24] L.J. Stockmeyer: "The polynomial time hierarchy". *Theoretical Computer Science* **3** (1977), 1–22.

[25] R. Szelepcsényi: "The method of forced enumeration for nondeterministic automata". *Acta Informatica* **26** (1988), 279–284.

[26] J. Torán: "Succinct representations of counting problems". 6th Int. Conference on Applied Algebra, Algebraic Algorithms, and Error Correcting Codes LNCS 357, Springer-Verlag (1988), 415–426.

[27] J. Torán: "Complexity classes defined by counting quantifiers". *Journal ACM* **38**, 753–774.

[28] K. Wagner: "The complexity of problems concerning graphs with regularities". Proc. Math. Found. of Comp. Sci. LNCS 176, Springer-Verlag (1984), 544–552.

[29] K. Wagner: "The complexity of combinatorial problems with succinct input representation". *Acta Informatica* **23** (1986), 325–356.