PH = PSPACE

Valerii Sopin

February 19, 2022

PH = PSPACE

Valerii Sopin[0000−0001−9953−6072]

VvS@myself.com, https://www.researchgate.net/profile/Valerii-Sopin

Abstract. We show that $\mathbb{PSPACE}$ is equal to 4th level in the polynomial hierarchy.

Keywords: Computational Complexity, Polynomial hierarchy, QBFs, PSPACE, BQP

1 Introduction

In computational complexity theory, $\mathbb{NP}$ is one of the most fundamental complexity classes. The complexity class $\mathbb{NP}$ is associated with computational problems having solutions that, once given, can be efficiently tested for validity. It is customary to define $\mathbb{NP}$ as the class of languages which can be recognized by a non-deterministic polynomial-time machine.

A decision problem is a member of co-$\mathbb{NP}$ if and only if its complement (the complement of a decision problem is the decision problem resulting from reversing the "yes" and "no" answers) is in the complexity class $\mathbb{NP}$. In simple terms, co-$\mathbb{NP}$ is the class of problems for which efficiently verifiable proofs of "no" instances, sometimes called counterexamples, exist. Equivalently, co-$\mathbb{NP}$ is the set of decision problems where the "no" instances can be accepted in polynomial time by a non-deterministic Turing machine.

On the other hand, $\mathbb{PSPACE}$ is the set of all decision problems that can be solved by a Turing machine using a polynomial amount of space.

An oracle machine is an abstract machine used to study decision problems. It can be visualized as a Turing machine with a black box, called an oracle, which is able to solve certain decision problems in a single operation. We use notation $\mathbb{L}^{\mathbb{O}}$, where $\mathbb{O}$ is the oracle.

On the contemporary state-of-the-art, the interested reader is referred to [1] and references therein.

Our result resolves some of unsolved problems in Computer Science. The idea of proof is the following:

1) the quantified Boolean formula problem is equivalent to existing of Boolean functions for each variable with the quantifier $\exists$, which make the formula to be tautology. A Boolean function doesn't depend on variables following after corresponding variable with the quantifier $\exists$.

2) size of Boolean functions can be exponential, but we consider it in frames of full DNF (each of its variables appears exactly once (in direct or inverse form) in every conjunction; that way, there is a one-to-one correspondence with truth table). It allows us to not construct precisely the Boolean functions. XOR is the only issue here and it can be handled.

The paper is organized as follows. Chapters 2-4 refresh basic definitions. Chapter 5 contains the proof.

2 Quantified Boolean formula

The Boolean Satisfiability Problem (abbreviated as SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values true or false in such a way that the formula evaluates to true.

SAT was the first known $\mathbb{NP}$-complete problem, as proved by Stephen Cook [2] and independently by Leonid Levin [3] (a problem is L-complete if it belongs to L and all problems in L have polynomial-time many-one reductions to it [1], L is the set of some decision problems).

One simple example of a co-$\mathbb{NP}$-complete problem is tautology, the problem of determining whether a given Boolean formula is a tautology; that is, whether every possible assignment of true/false values to variables yields a true statement.

For a Boolean formula $\phi(x_1, \ldots, x_n)$, we can think of its satisfiability as determining the true of the statement

$$\exists x_1 \in \{0,1\}\ \exists x_2 \in \{0,1\}\ \ldots\ \exists x_n \in \{0,1\}\ \ \phi(x_1, \ldots, x_n).$$

The SAT problem becomes more difficult if both "for all" ($\forall$) and "there exists" ($\exists$) quantifiers are allowed. It is known as the quantified Boolean formula problem or QSAT. QSAT is the canonical complete problem for $\mathbb{PSPACE}$ [1].

3 The Polynomial Hierarchy

We have seen the classes $\mathbb{NP}$ and co-$\mathbb{NP}$, which are defined as follows [1]:

$L \in \mathbb{NP}$ if there is a deterministic Turing machine M running in time polynomial in its first input, such that $x \in L \Leftrightarrow \exists w\ M(x; w) = 1$, w has length polynomial in x.

$L \in$ co-$\mathbb{NP}$ if there is a deterministic Turing machine M running in time polynomial in its first input, such that $x \in L \Leftrightarrow \forall w\ M(x; w) = 1$, w has length polynomial in x.

It is natural to generalize the above [1][4].

Let i be a positive integer. $L \in \Sigma_i$ if there is a deterministic Turing machine M running in time polynomial in its first input, such that

$$x \in L \Leftrightarrow \underbrace{\exists w_1 \forall w_2 \ldots Q_i w_i}_{i \text{ times}}\ M(x; w_1; \ldots; w_i) = 1,$$

where $Q_i = \forall$ if i is even, and $Q_i = \exists$ if i is odd.

Let i be a positive integer. $L \in \Pi_i$ if there is a deterministic Turing machine M running in time polynomial in its first input, such that

$$x \in L \Leftrightarrow \underbrace{\forall w_1 \exists w_2 \ldots Q_i w_i}_{i \text{ times}}\ M(x; w_1; \ldots; w_i) = 1,$$

where $Q_i = \forall$ if i is odd, and $Q_i = \exists$ if i is even.

As in the cases of $\mathbb{NP}$, co-$\mathbb{NP}$, we require that w_i each have length polynomial in x.

The polynomial hierarchy $\mathbb{PH}$ consists of all those languages of the form defined above. Note also the similarity to QSAT. The crucial difference is that QSAT allows an unbounded number of alternating quantifiers, whereas Σ_i, Π_i each allow (at most) i quantifiers. From here, $\mathbb{PH} \subseteq \mathbb{PSPACE}$.

4 Alternating Turing machine

An alternating Turing machine (ATM) is a non-deterministic Turing machine (NTM) with a rule for accepting computations that generalizes the rules used in the definition of the complexity classes $\mathbb{NP}$ and co-$\mathbb{NP}$. The concept of an ATM was set forth by Ashok Chandra, Larry Stockmeyer and Dexter Kozen [5].

The definition of $\mathbb{NP}$ uses the existential mode of computation: if any choice leads to an accepting state, then the whole computation accepts. The definition of co-$\mathbb{NP}$ uses the universal mode of computation: only if all choices lead to an accepting state, then the whole computation accepts. An alternating Turing machine (or to be more precise, the definition of acceptance for such a machine) alternates between these modes.

An alternating Turing machine with k alternations is an alternating Turing machine which switches from an existential to a universal state or vice versa no more than $k-1$ times. The complexity class $\mathbb{PH}$ is a special case of hierarchy of bounded alternating Turing machine [5].

$\mathbb{AP} = \mathbb{PSPACE}$, where $\mathbb{AP}$ is the class of problems alternating machines can solve in polynomial time [5].

5 Main result

Next theorem shows that QBF is indeed generalisation of the Boolean Satisfiability Problem, where determining of interpretation that satisfies a given Boolean formula is replaced by existence of Boolean functions that makes a given QBF to be tautology.

Theorem 1. *The quantified Boolean formula*

$$\Omega_1 x_1 \in \{0,1\}\ \Omega_2 x_2 \in \{0,1\}\ \ldots\ \Omega_n x_n \in \{0,1\}\ \ \phi(x_1, \ldots, x_n),$$

where $\phi(x_1, \ldots, x_n)$ is a Boolean formula, Ω_s, $s = i_1, \ldots, i_j$, is the quantifier $\exists$ and Ω_t, $t \neq i_1, \ldots, i_j$, is the quantifier $\forall$, j is the number of variables with the quantifier $\exists$, is a true quantified Boolean formula if and only if there are Boolean functions y_q, where y_q depends only on variables with the quantifier $\forall$ and indexes less i_q, $q = 1, \ldots, j$, that after substituting $x_{i_q} := y_q$ the given quantified Boolean formula becomes tautology.

Proof. It follows from simple recursive algorithm for determining whether a QBF is true. We take off the first quantifier and check both possible values for the first variable:

$$A = \Omega_2 x_2 \in \{0,1\} \ \dots \ \Omega_n x_n \in \{0,1\} \ \ \phi(0, \dots, x_n),$$

$$B = \Omega_2 x_2 \in \{0,1\} \ \dots \ \Omega_n x_n \in \{0,1\} \ \ \phi(1, \dots, x_n).$$

If $\Omega_1 = \exists$, then return A disjunction B (that's it, A or B is true; to avoid unambiguous, if A and B is true, take A for determining the function, so the value depends only on values of previous variables). If $\Omega_1 = \forall$, then return A conjunction B (A and B is true).

Notice that a Boolean function determines the truth table (one-to-one correspondence).

Example 1. Let only the quantifier for x_k, $k \geq 1$, be existential, then y_1 is some function of variables $x_1, \dots, x_{k-1}$, as QBF means in that case that for any possible values of $x_1, \dots, x_{k-1}$ there exists value of x_k that for all possible values of $x_{i>k}$ the given formula is true. It is indeed the truth table, where values of $x_1, \dots, x_{k-1}$ determine the value x_k.

Example 2. $\forall x_1 \exists z_1 \forall x_2 \exists z_2 \forall x_3 \exists z_3 \ \phi(x_1, z_1, x_2, z_2, x_3, z_3)$ is a true QBF if and only if there exist such Boolean functions $y_1 : \{0,1\} \to \{0,1\}$, $y_2 : \{0,1\}^2 \to \{0,1\}$, $y_3 : \{0,1\}^3 \to \{0,1\}$ that

$$\phi(x_1, y_1(x_1), x_2, y_2(x_1, x_2), x_3, y_3(x_1, x_2, x_3)) \text{ is tautology.}$$

Theorem 2.

$$\prod\nolimits_4 = (co\text{-}\mathbb{NP})^{\mathbb{NP}^{(co\text{-}\mathbb{NP})^{\mathbb{NP}}}} = \mathbb{PSPACE}$$

Proof. From [1][6] we know that without loss of generality we can assume a quantified Boolean formula to be in form (prenex normal form), where existential and universal quantifiers alternate. We assume it, for simplicity.

We wish that an quantified Boolean formula

$$\forall x_1 \in \{0,1\} \ \exists y_1 \in \{0,1\} \ \forall x_2 \in \{0,1\} \ \exists y_2 \in \{0,1\} \ \dots \ \forall x_n \in \{0,1\} \ \exists y_n \in \{0,1\}$$

$$\phi(x_1, y_1, \dots, x_n, y_n)$$

would be equivalent to
$\forall (x_1, x_2, \dots, x_n) \ \exists (y_1, \dots, y_n) \{$

$$\phi(x_1, y_1, x_2, y_2, \dots, x_n, y_n) \ \wedge$$

$$\wedge \ \forall (\hat{x}_n) \ \exists (z_n) \ \ \phi(x_1, y_1, x_2, y_2, \dots, x_{n-1}, y_{n-1}, \hat{x}_n, z_n) \ \wedge$$

$$\wedge \ \forall (\hat{x}_{n-1}, \hat{x}_n) \ \exists (z_{n-1}, z_n) \ \ \phi(x_1, y_1, x_2, y_2, \dots, x_{n-2}, y_{n-2}, \hat{x}_{n-1}, z_{n-1}, \hat{x}_n, z_n) \ \wedge \ \dots$$

$$\dots \ \wedge \ \forall (\hat{x}_2, \dots, \hat{x}_n) \ \exists (z_2, \dots, z_n) \ \ \phi(x_1, y_1, \hat{x}_2, z_2, \dots, \hat{x}_{n-2}, z_{n-2}, \hat{x}_{n-1}, z_{n-1}, \hat{x}_n, z_n)$$

$$\}$$

Namely, iterations of $\forall x \exists y$ reduce to conjunctions of separated $\forall \hat{x} \exists z$, as in the beginning we fix values of $\{y_q, \ q = 1, \dots, n\}$ and conjunctions jointly check that for predetermined $\{y_l, \ l < q\}$ suitable continuation $\{y_l, \ l \geq q\}$ can be found. In each conjunction we consider $\{y_l, l < q\}$ as functions dependent on all $\{x_i, \ i < q\}$ and $\{y_l, l \geq q\}$ as functions dependent on every $\{x_i, \ i = 1, \dots, n\}$ (if $\forall x_1 \ F(x_1, 0) = F(x_1, 1)$, then variable x_2 is dummy variable for Boolean formula $F(x_1, x_2)$). From here, if it is a true quantified Boolean formula, the above confirms it. However, another implication is not always true. Let's exam when two parts are different.

$\mathbf{n} = \mathbf{1}$**:** for a Boolean formula of one variable the equivalence obviously holds.

$\mathbf{n} = \mathbf{2}$**:** inconsistency can possibly happen only with $\exists y \ \forall x \ \phi(y, x)$; we have 16 different Boolean formulas of two variables and the equivalence is violated only for $XOR : (y \oplus x), \neg(y \oplus x)$.

$\mathbf{n} \geq \mathbf{3}$**:** taking off the first quantifier and checking both possible values for the first variable in way we did in Proposition, we come to the **n - 1** case. Indeed, for example, considering $\mathbf{n} = \mathbf{3}$, we have

$$\forall z \ \exists y \ \forall x \ \ \phi(z, y, x) \equiv \exists y \ \forall x \ \ \phi(0, y, x) \ AND \ \exists y \ \forall x \ \ \phi(1, y, x),$$

$$\exists t \ \forall x \ \exists y \ \ \phi(t, x, y) \equiv \forall x \ \exists y \ \ \phi(0, x, y) \ OR \ \forall x \ \exists y \ \ \phi(1, x, y),$$

where the second expression can be viewed as negation of the first expression. Consequently, it is enough to inspect only first expression due to double negation.

If $\exists y\, \forall x\;\; \phi(0, y, x) \equiv \forall x\, \exists y\;\; \phi(0, y, x)$ and $\exists y\, \forall x\;\; \phi(1, y, x) \equiv \forall x\, \exists y\;\; \phi(1, y, x)$, then $\forall z\, \exists y\, \forall x\;\; \phi(z, y, x) \equiv \forall z\; \forall x\; \exists y\;\; \phi(z, y, x) \equiv \forall z\; \exists \xi\; \forall x\; \exists y\;\; \phi(z, y, x)$. Otherwise, the equivalence is false due to *XOR* issue from $\mathbf{n = 2}$. Then $\exists y\, \forall x\;\; \phi(0, y, x)$ or $\exists y\, \forall x\;\; \phi(1, y, x)$ is false. Therefore, $\forall z\, \exists y\, \forall x\;\; \phi(z, y, x)$ is false.

Thus, using mathematical induction we have shown that XOR issue from $\mathbf{n = 2}$ appears whenever the equivalence we want doesn't work and the emergence means that the real value is false, but the displayed formula says that it is true. So, for each

$\forall(x_1, x_2, \ldots, x_n)\; \exists(y_1, \ldots, y_n)\; \forall(\hat{x}_i, \ldots, \hat{x}_n)\; \exists(z_i, \ldots, z_n)$

$$\phi(x_1, y_1, x_2, y_2, \ldots, \hat{x}_i, z_i, \ldots, \hat{x}_{n-1}, z_{n-1}, \hat{x}_n, z_n)$$

we additionally need to verify that $\phi(x_1, y_1, x_2, y_2, \ldots, \hat{x}_i, z_i, \ldots, \hat{x}_{n-1}, z_{n-1}, \hat{x}_n, z_n)$ as formula of two variables $(x \in \{x_1, \ldots, x_{i-1}, \hat{x}_i, \ldots, \hat{x}_n\}, y \in \{y_1, \ldots, y_{i-1}, z_i, \ldots, z_n\})$ (all other arguments are specified; there are $2n$ such formulas) is not equivalent to $\exists y\, \forall x\, (x \oplus y)$ or $\exists y\, \forall x\, \neg(x \oplus y)$. Otherwise, we consider the value ϕ with precise $x_1, y_1, x_2, y_2, \ldots, \hat{x}_i, z_i, \ldots, \hat{x}_{n-1}, z_{n-1}, \hat{x}_n, z_n$ as *false*, even if it is *true*. This extra condition can be checked in polynomial time.

To conclude, definition of alternating Turing machine shows that $(\text{co-}\mathbb{NP})^{\mathbb{NP}^{(\text{co-}\mathbb{NP})^{\mathbb{NP}}}}$ is enough and this way we solve complete problem for $\mathbb{PSPACE}$.

Remark 1. $\mathbb{PSPACE} = \mathbb{P}^{\mathbb{NP}}$? $\mathbb{PSPACE} = \mathbb{NP}^{\mathbb{NP}}$? $\mathbb{PSPACE} = \mathbb{NP}^{\mathbb{NP}^{\mathbb{NP}}}$?

$\mathbb{BQP}$ (bounded-error quantum polynomial time) is the class of decision problems solvable by a quantum computer in polynomial time, with an error probability of at most $1/3$ for all instances, see [1][6].

Corollary 1. *The polynomial hierarchy collapses and* $\mathbb{BQP} \subseteq \mathbb{PH}$.

Proof. See Chapter 3 and Theorem 2. It is known that $\mathbb{BQP} \subseteq \mathbb{PSPACE}$.

Remark 2. The relationship between $\mathbb{BQP}$ and $\mathbb{PH}$ has been a open problem since the earliest days of quantum computing [7].

Remark 3. In May 2018, Ran Raz and Avishay Tal published a paper [8] which showed that, relative to an oracle, $\mathbb{BQP}$ was not contained in $\mathbb{PH}$. However, note that an oracle separation of $\mathbb{BQP}$ and $\mathbb{PH}$ does not necessarily imply the ordinary separation. There is no contradiction.

$\mathbb{BPP}$ (bounded-error probabilistic polynomial time) is the class of decision problems solvable by a probabilistic Turing machine in polynomial time with an error probability bounded away from $1/3$ for all instances, see [1]. If the access to randomness is removed from the definition of $\mathbb{BPP}$, we get the complexity class $\mathbb{P}$.

Corollary 2. *If* $\mathbb{P} = \mathbb{NP}$*, then* $\mathbb{P} = \mathbb{PSPACE}$*. If* $\mathbb{BPP} = \mathbb{NP}$*, then* $\mathbb{BPP} = \mathbb{PSPACE}$*.*

Proof. If $\mathbb{P} = \mathbb{NP}$, then $\mathbb{NP} = \text{co-}\mathbb{NP}$, since $\mathbb{P} = \text{co-}\mathbb{P}$. Moreover, a $\mathbb{P}$ machine with the power to solve $\mathbb{P}$ problems instantly (a $\mathbb{P}$ oracle machine) is not any more powerful than the machine without this extra power. Thus, we obtain that $\mathbb{P} = \mathbb{PH}$.

$\mathbb{BPP}$ can be treated in the same manner, as it is known that $\mathbb{BPP}$ is closed under complement and low for itself, meaning that $\mathbb{BPP}^{\mathbb{BPP}} = \mathbb{BPP}$.

Corollary 3. *If* $\mathbb{NP} = $ *co-*$\mathbb{NP}$*, then* $\mathbb{NP} = \mathbb{PSPACE}$*.*

Proof. It is known that if $\mathbb{NP} = \text{co-}\mathbb{NP}$, then $\mathbb{NP} = \mathbb{PH}$.

$\mathbb{PP}$ is the class of decision problems solvable by a probabilistic Turing machine in polynomial time, with an error probability of less than $1/2$ for all instances, see [1][9]. $\mathbb{PP}$ has natural complete problems, for example, MAJSAT. It is a decision problem, in which one is given a Boolean formula ϕ. The answer must be "yes" if more than half of all assignments make ϕ true and "no" otherwise.

Corollary 4. $\mathbb{P}^{\mathbb{PP}} = \mathbb{PSPACE}$.

Proof. By Toda's theorem $\mathbb{PH} \subseteq \mathbb{P}^{\mathbb{PP}}$ [1][10]. Further, $\mathbb{P}^{\mathbb{PP}} \subseteq \mathbb{P}^{\mathbb{PSPACE}} = \mathbb{P}^{\mathbb{PH}} = \mathbb{PH}$.

Remark 4. By adding postselection to $\mathbb{BQP}$ ($\mathbb{BQP} \subseteq \mathbb{PP}$), a larger class is obtained [11]. It is known that it is equal to $\mathbb{PP}$ [11]. Is it true that $\mathbb{BQP} \neq \mathbb{PP}$?

Corollary 5. *If* $\mathbb{NP} \subseteq \mathbb{BQP}$*, then* $\mathbb{BQP} = \mathbb{PSPACE}$*.*

Proof. $\mathbb{BQP}$ is low for itself, which means $\mathbb{BQP}^{\mathbb{BQP}} = \mathbb{BQP}$ [12]; $\mathbb{BQP} \subseteq \mathbb{PSPACE}$.

Remark 5. Dependency quantified Boolean formulas (DQBFs) are a generalization of ordinary quantified Boolean formulas [13]. While the latter is restricted to linear dependencies of existential variables in the quantifier prefix, DQBFs allow arbitrary dependencies, which are explicitly specified in the formula. This makes decision problem with a DQBF to be $\mathbb{NEXP}$-complete (the complexity class $\mathbb{NEXP}$ is the set of decision problems that can be solved by a non-deterministic Turing machine using exponential time, i.e., in $O(2^{p(n)})$ time, $p(n)$ is a polynomial function of n) [14]. Theorem 2 is not applicable to the case of DQBFs directly as the looping is possible. Is it within reach to generalise Theorem 2 for it?

Remark 6. Theorem 2 opens the road for comprehensive pursuing of all exponential complexity classes and their relationships with probabilistic Turing machines and the polynomial hierarchy. The beginning of such kind of research can be found in [15][16][17][18][19].

Multiset $\{\mathbb{P}, \mathbb{NP}, \mathbb{NP}^{\mathbb{NP}}, \mathbb{NP}^{\mathbb{NP}^{\mathbb{NP}}}, \mathbb{NP}^{\mathbb{NP}^{\mathbb{NP}^{\mathbb{NP}}}}\}$ **shows that there is always a key.**

6 Acknowledgments

The author would like to thank Professors Lew Gordeew [20][21], Zhaohui Wei, Emil Jeřábek and the anonymous reviewers for their many comments and suggestions.

References

1. S. Arora and B. Barak, *Computational Complexity: A Modern Approach*, Cambridge University Press, 2009.
2. S. Cook, *The Complexity of Theorem-Proving Procedures*, Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 1971, 151 -158.
3. L. Levin, *Universal search problems*, Problems of Information Transmission, **9**:3, 1973, 115-116.
4. L. Stockmeyer, *The polynomial-time hierarchy*, Theoretical Computer Science, **3**, 1977, 1-22.
5. A. Chandra, D. Kozen, L. Stockmeyer, *Universal search problems*, Journal of the ACM, **28**:1, 1981, 114-133.
6. R. Jain, J. Ji, S. Upadhyay, *QIP = PSPACE*, Proceedings of the 42nd ACM symposium on Theory of computing, 2010, 573–582.
7. S. Aaronson, *BQP and the Polynomial Hierarchy*, Proceedings of the 42nd Annual ACM Symposium on Theory of computing, 2010, 141–150.
8. R. Raz, A. Tal, *Oracle Separation of BQP and PH*, https://eccc.weizmann.ac.il/report/2018/107/, 2018.
9. J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM Journal on Computing, **6**:4, 1997, 675–695.
10. S. Toda, *PP is as hard as the polynomial-time hierarchy*, SIAM Journal on Computing, **20**:5, 1991, 865–877.
11. S. Aaronson, *Quantum computing, postselection, and probabilistic polynomial-time*, Proceedings of the Royal Society A.,**461**:2063, 2005, 3473–3482.
12. E. Bernstein, U. Vazirani, *Quantum Complexity Theory*, SIAM Journal on Computing, **26**:5, 1997, 1411–1473.
13. G. Peterson, J. Reif, *Multiple-person alternation*, 20th Annual Symposium on Foundations of Computer Science, 1979, 348–363.
14. G. Peterson, J. Reif, S. Azhar, *Lower bounds for multiplayer non-cooperative games of incomplete information*, Computers and Mathematics with Applications, **41**:7-8, 2011, 957–992.
15. L. Babai, L. Fortnow, N. Nisan and A. Wigderson, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, Computational Complexity, **3**, 1993, 307–318.
16. R. Impagliazzo and A. Wigderson, *P = BPP if E requires exponential circuits: Derandomizing the XOR Lemma*, Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, 1997, 220–229.
17. S. Mocas, *Separating classes in the exponential-time hierarchy from classes in PH*, Theoretical Computer Science, **158**, 1996, 221–231.
18. D. Walther, C. Lutz, F. Wolter, M. Wooldridge, *Atl satisfiability is indeed ExpTime-complete*, Journal of Logic and Computation, **16**, 2006, 765–787.
19. S. Schewe, *ATL* Satisfiability Is 2EXPTIME-Complete*, International Colloquium on Automata, Languages, and Programming, **5126**, 2008, 373–385.
20. L. Gordeew, E. H. Haeusler, *Proof Compression and NP Versus PSPACE*, Studia Logica, **107**:1, 2019, 55–83;
21. L. Gordeew, E. H. Haeusler, *Proof Compression and NP Versus PSPACE II*, Bulletin of the Section of Logic, **49**:3, 2020, 213–230.