Electronic Colloquium on Computational Complexity, Report No. 134 (2022)

Some Games on Turing Machines and Power from Random Strings

Alexey Milovanov[1,2][0000−0002−4609−7851] and Greg McLellan

[1] HSE University, Russian Federation
[2] almas239@gmail.com
https://solid-lelik.jimdofree.com

Abstract. Denote by R the set of strings with high Kolmogorov complexity. In [3] the idea of using R as an oracle for resource-bounded computation models was presented. This idea was later developed in [1–3, 5, 6]. We prove new lower bounds for QP^R_{tt} and QP^R_{sa}:

- Oblivious NP $\subseteq \mathrm{QP}^R_{tt}$;
- Oblivious MA $\subseteq \mathrm{QP}^R_{sa}$.

Here QP means quazi-polynomial-time; "sa" means sub-adaptive reduction—a new type of reduction that we introduce. This type of reduction is not weaker than truth-table reduction and is not stronger than Turing reduction.

Also we prove upper bounds for BPP^R_{tt} and P^R_{sa} following [1]:

- $\mathrm{P}^R_{sa} \subseteq \mathrm{EXP}$;
- $\mathrm{BPP}^R_{tt} \subseteq \mathrm{AEXP}^{\mathrm{poly}}$.

Here $\mathrm{AEXP}^{\mathrm{poly}}$ is the class of languages decidable in exponential time by an alternating Turing machine that switches from an existential to a universal state or vice versa at most polynomial times.

Finally we analyze some games that originate in [1]. We prove completeness of these games. These results show that methods in [1] can not prove better upper bounds for P^R, NP^R and P^R_{tt} than known.

Keywords: Kolmogorov complexity · Complexity classes · Games.

ISSN 1433-8092

Introduction

Denote by $\mathrm{K}_U(x)$ the prefix complexity of x with respect to a universal decompressor U—the minimal length of a prefix-free program that outputs x. (For the definition and properties of Kolmogorov complexity see for example [10] or [8].) Denote by R_U the oracle function that outputs $\mathrm{K}_U(x)$ on input x. In [1–3, 5, 6] the following results are proved:

$$\mathrm{EXP}^{\mathrm{NP}} \subseteq \bigcap_U \mathrm{P}^{R_U}, \bigcap_U \mathrm{NP}^{R_U} \subseteq \mathrm{EXPSPACE},$$

$$\mathrm{BPP} \subseteq \bigcap_U \mathrm{P}_{tt}^{R_U} \subseteq \mathrm{PSPACE},$$

$$\mathrm{NEXP} \subseteq \bigcap_U \mathrm{BPP}_{tt}^{R_U} \subseteq \mathrm{EXPSPACE}.$$

$$\mathrm{H} \in \bigcap_U \mathrm{P}^{R_U}/\mathrm{poly}.$$

Here H is a Halting problem; P_{tt}^A is $\{L : L \leq_{tt}^{\mathrm{P}} A\}$, BPP_{tt}^A is $\{L : L \leq_{tt}^{\mathrm{BPP}} A\}$; the intersection (everywhere) is by all universal decompressors U. [3]

For convenience introduce the abbreviations: $\mathrm{NP}^R := \bigcap_U \mathrm{NP}^{R_U}$,
$\mathrm{P}^R := \bigcap_U \mathrm{P}^{R_U}$, $\mathrm{P}_{tt}^R := \bigcap_U \mathrm{P}_{tt}^{R_U}$, $\mathrm{BPP}_{tt}^R := \bigcap_U \mathrm{BPP}_{tt}^{R_U}$.

In Section 1 we prove a lower bound for $\mathrm{QP}_{tt}^{R_U}$. Here $\mathrm{QP} = \bigcup_i \mathrm{Time} 2^{O(\log^i n)}$. Recall the definition of Oblivious NP.

Definition 1. *A language L is in Oblivious NP if there exists a polynomial time verifier V taking an input and a witness, so that: there is a witness for each n of polynomial size, so that for any input of size n,*

- *if the input is in L, then the verifier accepts on that input and the witness.*
- *If the input is not in L, then for any witness, the verifier rejects on that input.*

We prove the following theorem.

Theorem 1.

$$\textit{Oblivious NP} \subseteq \mathit{QP}_{tt}^{R}.$$

Introduce new types of reductions.

Machine M with an oracle access defines a reduction tree—see Figure 1.

Definition 2. *A machine M with an oracle access is called* strictly sub-adaptive *if for every input string x all nodes in the reduction tree (all the oracle requests) are different—see Figure 1.*

The following type of reduction is of greater interest. This is a "mix" of tt and strictly sub-adaptivity reductions.

[3] In [2, 3] the plain complexity instead of prefix complexity was considered. However, as was mentioned in [1] this change does not affect on the lower bounds for $\bigcap_U \mathrm{NP}^{R_U}$, $\bigcap_U \mathrm{P}^{R_U}$ and $\bigcap_U \mathrm{P}_{tt}^{R_U}$.

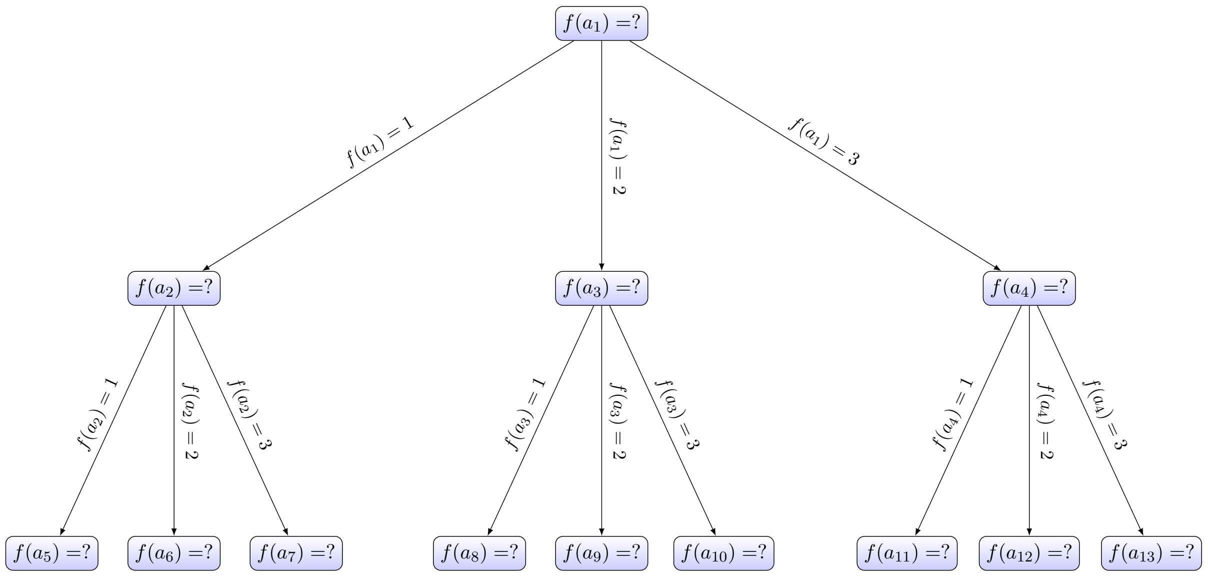


Fig. 1. For strictly sub-adaptivity reduction all nodes are different

Definition 3. *A machine M with an oracle access is called* sub-adaptive *if for every input string x the following holds. The computation of $M(x)$ with an oracle consists of two steps. In the first step M asks non-adaptively several requires to an oracle. In the second step M works strictly sub-adaptivity. Moreover, in the second step all oracle requests are different—see Figure 2.*

This type of reduction is not stronger than Turing reduction and is not weaker than tt-reduction. Denote by $\mathrm{P}_{sa}^{R_U}$ the class of languages that are recognized in polynomial time by a sub-adaptive Turing machine using oracle function R_U. Denote

$$\mathrm{QP}_{sa}^{R} := \bigcap_{\text{universal } U} \mathrm{QP}_{sa}^{R_U}.$$

For QP_{sa}^{R} we get some better lower bound than QP_{tt}^{R}— Oblivious MA.

Definition 4. *A language L is in Oblivious MA if there exists a randomized, polynomial time verifier V taking an input and a witness, so that:*

There is a witness for each n of polynomial size, so that for any input of size n,

- *if the input is is in L, then the verifier accepts on that input and the witness.*
- *If the input is not in L, then for any witness, the verifier rejects on that input with probability at least $\frac{1}{2}$.*

Theorem 2.

$$\textit{Oblivious MA} \subseteq QP_{sa}^{R}.$$

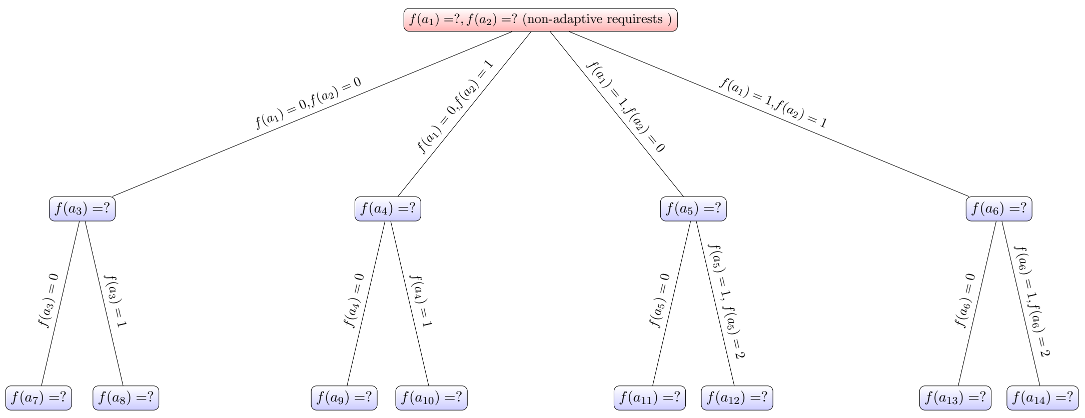


Fig. 2. For sub-adaptive reduction at the second step all nodes are different.

In Section 2 we give a high-level outline of the proofs upper bounds for NP^R, P^R and P^R_{tt} from [1]. The idea is to reduce the problem to some finite game between two players. The problem of finding a winning player belongs to EXPSPACE (for NP^R and P^R) and to PSPACE (for P^R_{tt}). This fact (by some reasons) means that $\mathrm{NP}^R, \mathrm{P}^R \subseteq$ EXPSPACE and $\mathrm{P}^R_{tt} \subseteq$ PSPACE.

In Section 4 we prove that these games are complete for the corresponding classes. This means that methods in [1] can not prove better upper bounds for P^R, NP^R and P^R_{tt}.

In Section 3 we analyze similar games in context sub-adaptive and randomize tt-reductions. We prove the following theorems.

Theorem 3.
$$P^R_{sa} \subseteq EXP.$$

Theorem 4. *$BPP^R_{tt} \subseteq AEXP^{poly}$.*

Here $\mathrm{AEXP}^{\mathrm{poly}}$ is the class of languages decidable in exponential time by an alternating Turing machine that switches from an existential to a universal state or vice versa at most polynomial times.

In Subsection 4.3 we discuss sub-adaptivity in the context of NP-Turing reductions.

1 Lower bounds for QP^R_{tt} and QP^R_{sa}

Fix some universal U and denote $\mathrm{K}(x) = \mathrm{K}_U(x)$ and $R = R_U$.[4]

The key ingredient for proving Theorems 1 and 2 is the following fact.

[4] The results of this section are valid for other types of Kolmogorov complexity (plane complexity, monotone complexity,...) that are equal each other with logarithmic precision.

Theorem 5. *For every C there exists a quazi-polynomial in n algorithm using non-adaptively oracle-function R that on input n outputs a quazi-polynomial size list A of strings that contains all strings with length n and Kolmogorov complexity at most $C \log n$.*

Proof. Consider a string x of length n as the truth-table of a function $f_x : \{0,1\}^{\log n} \to \{0,1\}$. In [2] the following connection between $\mathrm{K}(x)$ and circuit-complexity of f_x is proved.

Theorem 6 ([2]). *For every n and for every string x the function f_x can be computed by a circuit of size $\mathrm{poly}(\mathrm{K}(x) + \log n)$ with access to Halting problem H as oracle.*

Recall that Halting problem can be computed by poly-size circuits with oracle R. Hence, for every x with length n and complexity $O(\log n)$ the function f_x can be computed by a $\mathrm{poly}(\log n)$-size circuit with oracle R.

The required algorithm on input n constructs all $\mathrm{poly}(\log n)$-size circuit with oracle R and computes all truth tables of the resulting functions. Note that all requires to oracle R in this algorithm has length $\mathrm{poly}(\log n)$, so the requires can be done non-adaptively.

Proof (Proof of Theorem 1). Let L be a string from Oblivious NP. Denote by y the witness for strings of length n. Note that Kolmogorov complexity of y is equal to $O(\log n)$ since y is restored from n in a computable way.

Let x be an input. Run the algorithm from Theorem 5 for $n = |y|$ (the length of y). Since $|y| = \mathrm{poly}(|x|)$ this algorithm works quasi-polynomial in $|x|$ time. Getting the required list of simple strings it remains to find there a witness if such exists.

Proof (Proof of Theorem 2). The first step of the proof is the same as in the proof above. It remains to derandomize the verifier. The idea is the same as in the proof of $\mathrm{P}^R = \mathrm{BPP}^R$ [2]. The algorithm construct (using oracle R sub-adaptively) a string r with poly-logarithmic length such that $\mathrm{K}(r) = \Omega(|r|)$. After this we use f_r as a hard-function in pseudo-random generator from [2, 7] and derandomize the verifier.

Give more detailed proof. First we get the list of strings with simple complexity from Theorem 5. For every string from the list do the following. Denote by m a binary string that code the previous oracle requires, and the oracle answers. First we ask $\mathrm{K}(m)$. Our further requires are continuations of word m. It guarantees that in the reduction tree nodes from different sub-trees (that corresponds to different values of m) are not intersected.

Now our goal is to construct a string r of length $2^{\mathrm{poly}(\log n)}$ (here n is the length of input) such that

$$\mathrm{K}(mr) \geq \mathrm{K}(m) + \frac{|r|}{2}. \qquad (1)$$

For this we use the following well-known fact in Kolmogorov complexity theory.

Lemma 1 ([10]). *For every string z and for every m there exists a string y of length m such that* $\mathrm{K}(zy) \geq \mathrm{K}(z) + |m| - O(\log m)$.

From this lemma it follows that we can add to m a suffix of logarithmic length such that the condition (1) holds (as it was done in [2]). To safe sub-adaptivity we find the lexicographical first such suffix.

From 1 it follows that $\mathrm{K}(r) \geq |r|/2 - O(\log |r|)$. Recall that $|r| = 2^{\mathrm{poly}(\log n)}$. We argue (as it was mentioned in [2]) that the function f_r can not be computed by circuits with oracle R of size less than $|r|^{\varepsilon}$ for some positive ε. For oracle H instead of R it immediately follows from the following theorem.

Theorem 7. *There is a $\delta > 0$ such that For every string r the minimal size of a circuit with oracle H computing f_r is at least* $\mathrm{K}(r)^{\delta}$.

It is not difficult to see that R can be computed by poly-size circuits with oracle H—see [2]. Hence, f_r can not be computed by small circuits with oracle R. To derandomize the verifier it remains to use the following pseudo-random generator.

Theorem 8. *For any $\varepsilon > 0$, there exist constants $c, c_0 > 0$ such that the following holds. Let A be a set and $l > 1$ be an integer. Let $f : \{0,1\}^{c \log l} \to \{0,1\}$ be a boolean function that cannot be computed by oracle circuits of size $l^{c\varepsilon}$ with oracle A. Then $G_f : \{0,1\}^{c_0 \log l} \to \{0,1\}^l$ satisfies:*

$$|\Pr_{r \in U_l}[C^A(r) = 1] - \Pr_{x \in U_{c_0 \log l}}[C^A(G_f(x)) = 1]| < 1/l,$$

for any oracle circuit C^A of size at most l.

To derandomize the verifier it is enough to use this theorem for $l = 2^{\mathrm{poly}(\log n)}$.

2 The idea of proof upper bounds for classes containing oracle *R*

Here we give the proof of the following theorem from [1].

Theorem 9. *Let L be a decidable language not in EXPSPACE. Then there exists an optimal prefix-free decompressor D such that L is not polynomially Turing reducible to the corresponding complexity function* $\mathrm{K}(\cdot) = \mathrm{K}_D(\cdot)$.

In [5] it was shown that P^R and NP^R contain only decidable languages. Together with Theorem 9 it gives that $\mathrm{P}^R \subseteq \mathrm{EXPSPACE}$.

This theorem can be reformulated as a game. The main idea is that the $2^{-\mathrm{K}_U(x)}$ defines a universal semimeasure for suitable universal U.

Game reformulation Fix some decidable L not in EXPSPACE. Consider the following game with full information. Alice and Bob start with a function $k(x) = 2|x| + c$ where c is large enough constant such that $\sum 2^{-k(x)} \leq 1/4$. At any moment each of the players may decrease the value of the function (replacing this value by some smaller non-negative integer). The cost of this decrease is the increase in $\sum_x 2^{-k(x)}$. There is a total budget for Alice—1/2 and for Bob—1/4; moves that violate the budget restriction are illegal.

The game is infinite; the winner is determined by the limit of the decreasing function k. Namely, Bob wins if L is polynomially Turing reducible to the limit function k, otherwise Alice wins.

Remark 1. We may assume that players alternate or use any other ordering (where each player makes infinitely many moves), since the winner is determined by a limit function and each player may postpone his/her move (this may only save the player's money and make the opponent spend more).

Lemma 2. *If Alice has a computable winning strategy in this game, then the statement of Theorem 9 is true.*

Proof. Let Alice play against the blind strategy of Bob that decreases the values of $k(x)$ until $\mathrm{K}(x) + 2$ (the constant 2 is needed not to exceed the budget of 1/4). This strategy is computable, so we get a computable sequence of moves if Alice uses her computable strategy against this blind strategy. Therefore the limit function will be upper semi-computable, the corresponding series has sum at most 1 (since the initial function had sum at most 1/4, and players increase it at most by 3/4), and the limit function is optimal since Bob guarantees this. On the other hand, there is no reduction since Alice wins in the game.

A series of requirements The winning condition in the game is a conjunction of a countable series of requirements: for each machine M there is a requirement "M does not perform a truth table polynomial reduction of L to the limit function k". We can effectively enumerate these requirements: we assume that for each machine some bounding polynomial is declared (as part of its description), and stop the computation when the time bound is reached (giving an arbitrary output). Let M_i be the sequence of the bounded machines; the corresponding requirement "M_i does not reduce L to k" we will also denote by M_i.

How to deal with one requirement For a start, let us consider the simplified case when we have only one requirement (machine) M. Why can Alice win the game in this case? To win the game, Alice should ensure that for some x the output of the reduction machine M on x is different from the true answer $L(x)$ (`true` if x in L, `false` otherwise). Consider some x and let us see what Alice can do to destroy the reduction at this x. The pair M and x defines a reduction tree (see Figure 1). Since M is a polynomial-time machine the size of the tree is exponential in the length of x. So, there are exponential many strings a_1, a_2,... a_n whose limit values of $k(a_1), \ldots, k(a_n)$ define the output $M(x)$.

We may consider a "local" version of the game that deals only with $a_1, \ldots, a_n$. Initially we have a list of n numbers $(2|a_1|+c, \ldots, 2|a_n|+c)$, Alice and Bob may at any time decrease any of these numbers (paying the corresponding amount, the increase in $2^{-k(a_i)}$), each not exceeding his/her budget $1/4$. Alice wins if she may ensure that the reduction answer for the limit values is different from $L(x)$. If Alice can win in such a game for some x, this is enough: using the winning strategy for the local game, she wins the global game. (She never decreases other values of k, and if Bob does so, he just wastes his budget, since only the values $k(a_i)$ matter.)

We arrive at the main point of the argument where the class EXPSPACE appears: for each x we have a local game between two players who have symmetric rights. For such a game, *either there is a strategy which guarantees that M outputs* 0*, or there is a strategy that guarantees that M outputs* 1. (We do not say who uses this strategy, since the rules of the game are symmetric, only the winning condition is different for Alice and Bob.) *And one can decide in EXPSPACE which of the two cases happens for a given x.*

Lemma 3. *The language containing all strings x such that there is a winning strategy to get $M(x) = 1$ belongs to EXPSPACE.*

The same is true if we change $1/4$ (budget of players) to any other positive constant.

Proof. Recall that the number of strings a_1, a_2, $\ldots$, a_n is at most exponential in the length of x. Therefore, the total number of steps in this game is also bounded by some exponent. (A player can miss his or her step. However, we assume that a player can do it only if they find the current value of $M(x)$ acceptable.) Hence, this game belongs to EXPSPACE by a standard argument.

Since L is *not* in EXPSPACE (by assumption), there is some x for which $L(x)$ is *not* equal to the answer that can be enforced by a strategy. For this x Alice can enforce the answer that is opposite to what L says, so she can win the local game (and the global one, if there were only one requirement). Since L is decidable Alice can effectively find such an x.

We complete the proof of Theorem 9 in the Appendix (alternatively, a reader can read [1]). The statement $\mathrm{P}_{tt}^{R} \subseteq$ PSPACE has the same proof: one need only use machines that implement truth-table reduction. For these machines the corresponding game belongs to PSPACE.

Moreover, we can state the following general statement. Let $\mathcal{M}$ be some enumerable family of Turing machines (for example, polynomial-time Turing machines) with access to some oracle-function. Consider the local game for machines in this family as in Lemma 13. Assume that the language containing all strings x such that there a is winning strategy to get $M(x) = 1$ belongs to some complexity class $\mathcal{C}$. Then we state the following theorem.

Theorem 10. *Let L be a decidable language. Assume that for every universal U language L is decidable by some machine $M \in \mathcal{M}$ with oracle R_U. Then L belongs to $\mathcal{C}$.*

We affirm that this theorem has the same proof as Theorem 9.

3 Upper bounds for P_{sa}^{R} and P_{tt}^{R}

Prove that $\mathrm{P}_{sa}^{R} \subseteq \mathrm{EXP}$. using the technique from the previous section.

Proof (Proof of Theorem 3). We know that $\mathrm{P}_{sa}^{R} \subseteq \mathrm{P}^{R} \subseteq \mathrm{EXPSPACE}$, so every language in P_{sa}^{R} is decidable. Hence, due to Theorem 10 it is enough to prove that the corresponding local game for sub-adaptive reduction belongs to EXP. There is, however, a subtle point. There is no algorithm that can determine if a given machine computes a sub-adaptive reduction. So we can not effectively enumerate them. However this is not a problem. Instead of enumerating sub-adaptive polynomial-time machines Alice can enumerate all polynomial-time machines. For every machine M Alice can either find x $M^{R}(x) \longleftrightarrow x \notin L$ (for some language $L \notin \mathrm{EXP}$) or figure that M is not sub-adaptive.

First we explore local game for strong sub-adaptivitely. Note that if a player has a winning position then they can omit their move (it is not worse then making any other move). Hence, we can change the rules—players in winning position are not allowed to move. The current position can be described by a path in a reduction tree—see Figure 3. A player (with a losing position) must change the current value of f on the current path of the reduction tree. (In Figure 3 it is not possible to change $f(a_1)$ or $f(a_3)$ (the value $f(a_{10})$ is minimal so it is possible to change this value)). It is possible that a player has to make several steps successively to change the value of output.

Now we will use sub-adaptivity. The key note is that vertices that are to the left of the current path will be not change (such moves do not have sense so we can assume that players do not make such moves). Note that sub-adaptivity is crucial: in general case this is not true (and the corresponding game is EXPSPASE-complete by Theorem 12).

So, the current game position is completely specified by triple (the current path, Alice's current capital, Bob's current capital). We claim that the number of such triples is an is exponential in the size of the input. Indeed, the tree is exponentially large. The capitals are binary-rational numbers with denominator $2^{\max}$, where max is the maximal number among all numbers $f(y)$ that the machine asks. Since $f(y) \leq 2|y| + c$ and all lengths are bounded by poly(input size) the same is true for max.

Therefore, it is possible to find a winner in exponential time by using dynamic programming.

For general sub-adaptively the reasoning is the same: since at the first stage a machine asks polynomially oracle requests non-adaptively the number of possible game-positions is exponentially large for general sub-adaptively too.

Now we prove that $\mathrm{BPP}_{tt}^{R} \subseteq \mathrm{AEXP}^{\mathrm{poly}}$

Proof (Proof of Theorem 4). Let $L \in \mathrm{BPP}_{tt}^{R}$. Since $\mathrm{BPP}_{tt}^{R} \subseteq \mathrm{BPP}^{R} = \mathrm{P}^{R} \subseteq \mathrm{EXPSPACE}$ the language L is decidable. So, we can use the same technique. As in the previous theorem we can not enumerate all polynomial-time machines that provide a randomize tt-reduction. The problem is that for given randomize

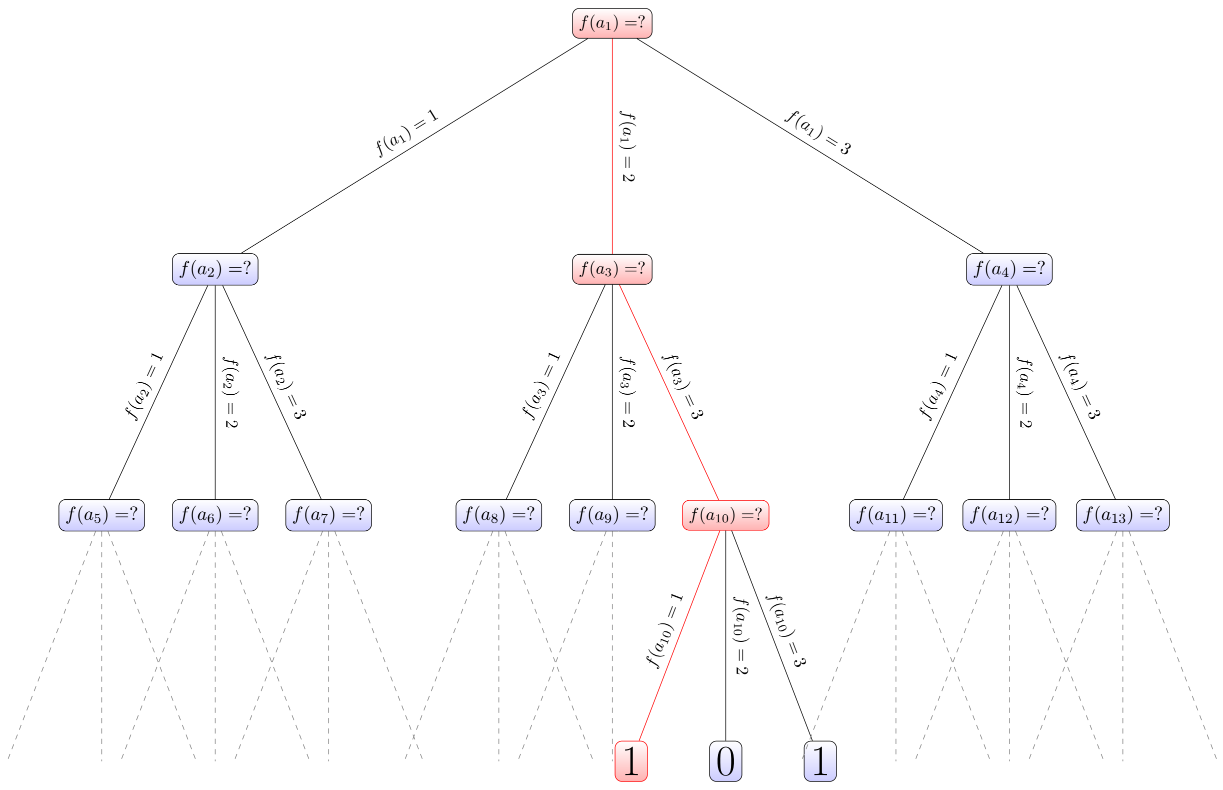


Fig. 3. The current values of oracle f: $(f(a_1) = 2, f(a_3) = 3,\ f(a_{10}) = 1)$ defines the reduction path. This path defines the output of the machine (here it is 1).

machine M and oracle A the values $\Pr[M^A(x) = 1]$ and $\Pr[M^A(x) = 0]$ depend on A.

But Alice can enumerate all randomize polynomial-time oracle machines that use this oracle non-adaptively. As in the previous proof for every such machine M she either finds x such that $M^R(x) \longleftrightarrow x \notin L'$ for $L' \notin \mathrm{AEXP}^{\mathrm{poly}}$ or figure that M^R is not provide a randomize tt-reduction.

Note that by an amplification argument if $L \in \mathrm{BPP}^R_{tt}$ then there is a corresponding machine M such that:

- if $x \in L$ then $\Pr[M^R(x) = 1] > 1 - 2^{-|x|}$;
- if $x \notin L$ then $\Pr[M^R(x) = 1] < 2^{-|x|}$.

Therefore, to complete the proof it is enough to show that the following local BPP_{tt}-game belongs to $\mathrm{AEXP}^{\mathrm{poly}}$.

Local game for $\mathbf{BPP}_{tt}$ There are a polynomial-time randomize oracle machine M^A that uses oracle A non-adaptively and its input x. Initially the value

$\Pr[M^A(x) = 1]$ is greater than $1 - 2^{-|x|}$ or smaller than $2^{-|x|}$. There are two players with some budgets that can change values of A by using their money. Every value can be changed $q(n)$ times only (here n is the length of x and q is some polynomial).

The first player wants to $\Pr[M^A(x) = 1] > 1 - 2^{-|x|}$ and second player wants to $\Pr[M^A(x) = 1] < 2^{-|x|}$. One (big) move of a player is changing A to make the value $\Pr[M^A(x) = 1]$ "good" for a player.

Prove that BPP_{tt}-game belongs to $\mathrm{AEXP}^{\mathrm{poly}}$. For this it is enough to show the number of (big) moves in this game is bounded by $\mathrm{poly}(n)$. Indeed, every big move is described by choosing exponentially many oracle requests.

Choose a polynomial p and show that this game can not continue more than $p(n)$ moves if p is large enough. Consider M as a deterministic machine depending on its input x and random bits r. Note that for every fixing r a machine $M^A(x, r)$ can make at most $\mathrm{poly}(n)$ oracle requests; denote this polynomial as $h(n)$.

The game position is depending on fraction of r such that $M^A(x, r) = 1$. After every big move for at most every r the value $M^A(x, r)$ is changed: after the first big move the fraction of such r is at least $1 - 2^{-|x|+1}$; after $p(n)$ moves the value $M^A(x, r)$ is changed $p(n)$ for $1 - 2^{-|x|+1}p(n)$-fraction of r. However, the value of $M^A(x, r)$ can not be changed more than $h(n)q(n)$ times (recall that $h(n)$ is an upper bound for oracle requests for fixing r and $q(n)$ is an upper bound for possible changing an oracle value by players). So, if $p(n) > h(n)q(n)$ then the number of big moves in this game is bounded by $p(n)$.

4 Completeness of some games

4.1 Game for tt-reduction

Here we consider the game like a local game in Lemma 13 but for tt-reduction.

Let M be a polynomial time Turing machine having access to oracle O. This machine implements tt-reduction, i.e. on inputs of length n machine M asks $\mathrm{poly}(n)$ YES/NO-question to oracle O. After this machine outputs 1 or 0. Initially O is empty. Let x be some string.

Consider the following game. The goal of Alice is $M^O(x) = 1$, the goal of Bob is $M^O(x) = 0$. Alice and Bob can add strings to O for some money. Namely, adding string y costs $v(y)$, where v is some polynomial time computable function $\{0, 1\}^* \to \mathbb{N}$. The players moves alternate but they can omit their moves if the current value $M^O(x)$ is OK for him or for her. Initially Alice has c_A dollars, Bob has c_B dollars.

Theorem 11. *There exists a polynomial-time Turing machine M that realizes tt-reduction and a positive polynomial computable function v such that the following is true. The language*

tt-GAME$= \{(x, c_A, c_B)|$ Alice has a winning strategy in the game described above $\}$ is PSPACE-complete.

The proof of this theorem is in Appendix. Note that here we consider an oracle that outputs YES/NO question instead of oracle-function. It is clear that completeness for a binary oracle implies completeness in the general case.

4.2 Game for Turing reduction

In this subsection we consider the similar game as in the previous subsection with the following difference: here we consider polynomial time Turing machines realizing Turing reductions instead of tt-reductions.

Theorem 12. *There exists a polynomial-time Turing machine M that realizes Turing reduction and a positive polynomial computable function v such the following is true. The language*

T-GAME= $\{(x, c_A, c_B)|$ Alice has a winning strategy in the game described above $\}$ is EXPSPACE-complete.

The proof of this theorem is in Appendix.

4.3 Non-deterministic sub-adaptivity

Define class NP^R_{sa} by the following way. A non-deterministic polynomial Turing machine with an oracle-function defines exponentially-many trees of reduction (corresponding to different paths of the machine). A non-deterministic Turing machine is called sub-adaptive if all queries in all of the trees are different. The class of all languages that can be recognized by sub-adaptive polynomial-time non-deterministic machines with oracle R_U denotes as $\mathrm{NP}^{R_U}_{sa}$. As for P we define NP^R_{sa} as $\bigcap_{\text{universal } U} \mathrm{NP}^{R_U}_{sa}$.

Clearly, $\mathrm{NP}^R_{sa} \subseteq \mathrm{NP}^R \subseteq$ EXPSPACE, but can we get a better upper bound? In this subsection we consider a similar game as in the previous subsection but for $\mathrm{NP}^{R_U}_{sa}$.

First, consider the following generalization of the game for $\mathrm{P}^{R_U}_{sa}$. There is a directed weighted graph G with a chip at some node. All nodes of G are marked by A or B. There are two players Alice and Bob. The goal of Alice (Bob) is to shift the chip to a node that is marked by A (B). Initially Alice and Bob have m_A and m_B dollars respectively. If a player is in a losing position (i.e., current position of the chip is marked by opposite letter) he or she can move the chip to a neighboring node. Such move costs some dollars (the weight of the corresponding edge).The player loses if he or she is in a losing position and does not have enough money to fix it. Now consider the language GRAPH-GAME that consists of all directed weighted graphs G (all weights are positive integers), initial position of the chip, and capitals of Alice and Bob that are given in the unary representation such that Alice has a wining strategy at this game.

The language GRAPH-GAME belongs to P. Indeed, the current position of the game is defined by the position of the chip and the current capitals of Alice and Bob, so dynamic programming works (here it is important that initial capitals are given in the unary representation). We claim that this is a generalization of the game in the previous subsection (note, that the size of a graph (a tree) in the previous subsection was exponential, this is why we get there EXP, not P).

What kind of game corresponds to NP^{R}_{sa}? Instead of one graph we need to consider several directed weighted graphs $G_1, \ldots G_n$ with a chip at each graph. All nodes of all graphs are marked by A and B. Now Bob wins if all chips are marked by B and Alice wins if at least one chip is marked by A.

Consider the language GENERAL-GRAPH-GAME that consists of all graphs $G_1, \ldots, G_n$, initial positions and capitals m_A and m_B (in the unary representation) such that Alice wins at the corresponding game. Here it is important that capitals are common for all graphs so, it is not just several independent GRAPH-GAMEs.

Theorem 13. *The language GENERAL-GRAPH-GAME is PSPACE-complete.*

The proof of this theorem is also in Appendix.

References

1. E. Allender, L. Friedman, and W. Gasarch. Limits on the computational power of random strings. *Information and Computation*, 222:80-92, 2013.
2. Allender, E., Buhrman, H., Koucký, M.: What can be efficiently reduced to the Kolmogorov-random strings? *Annals of Pure and Applied Logic* 138, 2–19 (2006)
3. E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35:1467–1493, 2006.
4. L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.
5. M. Cai, R. Downey, R. Epstein, S. Lempp, and J. Miller. Random strings and tt-degrees of Turing complete c.e. sets. *Logical Methods in Computer Science*, 10(3):1–24, 2014
6. Shuichi Hirahara, Unexpected hardness results for Kolmogorov complexity under uniform reductions, Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020), 1038–1051.
7. A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31:1501–1526, 2002
8. Li M., Vitányi P., *An Introduction to Kolmogorov complexity and its applications*, 3rd ed., Springer, 2008 (1 ed., 1993; 2 ed., 1997), xxiii+790 pp. ISBN 978-0-387-49820-1.
9. D. Pálvölgyi, https://cstheory.stackexchange.com/questions/41762/what-is-the-complexity-of-ths-game
10. Shen, A., Uspensky V. and Vereshchagin N.: *Kolmogorov Complexity and Algorithmic Randomness*, ACM, (2017).

Appendix

Completion of proof of Theorem 9

Dealing with many requirements: the problem What are the problems if we try to use the argument explained above against *two* machines (to guarantee that none of them reduces L to k)? There are two problems. First, the sets of strings $a_1, \dots, a_k$ (the k-values used for both reductions) may intersect, and the change in one local game influences the other. Second, the budget in the global game is shared between the local games, so the strategies in the local games that assume a fixed budget do not work any more.

The solution is to deal with the machines M_i sequentially. Imagine we have two machines M_1 and M_2. First we deal with M_1 as if M_2 did not exist at all. When the game reaches the final stage, we start analyzing the local game for M_2, but use the current values of k as the initial configuration for M_2. It is possible since our argument (about symmetric games and EXPSPACE) does not depend on the initial values of k. The we can treat M_2 in the same way as before (for one machine).

There are two problems with this argument. First, we do not know "when the game reaches the final stage": the blind strategy played by Bob may decrease any value of k at any moment. This is not so bad, since we can restart the actions against the second machine many times (and hope each time that the first game is finished). In the same way one may treat countably many conditions (priority argument).[5]

The second problem, with the common budget, is more serious. It is possible that in the first game (against M_1) Alice wins but uses almost all bugdet (1/4 in our example) while Bob used only a small part of his budget (also 1/4 in our example). Then, if we use the remaining resources for the second game, it is no more symmetric (and is biased in the wrong direction), so the game argument cannot be used anymore. For two conditions we may allow Alice to use 1/4 in the first game and (separately) 1/4 in the second game, while Bob is allowed to use only 1/4 in total, and this almost saves the argument: it is OK that Alice uses $1/4 + 1/4 = 1/2$ in total. The only remaining problem is that the second game is restarted many times (when a new move is made by Bob in the first game), and if 1/4 is allocated for each restart, then the total spending of Alice is unlimited. What can be done?

An economical way to deal with one requirement To solve these problems, we modify the argument for one requirement. Assume that some small ε is chosen, and we consider a symmetric local game where both players are allowed to spend at most ε. Alice chooses some x where the EXPSPACE-prediction of the outcome of this ε-bounded game differs from L, and uses the winning strategy to play with Bob. However, Bob does not know that his spending is bounded by ε, and may — instead of losing the game — spend more. Then the game is abandoned, and new game (again with symmetric bound ε, with new x where L differs from EXPSPACE-set) is started. The main advantage of this approach

[5] Imagine a knight who deals with infinitely many dragons; each dragon has only finitely many lives (but the number is unknown); moreover, when the knight hits ith dragon, all subsequent dragons are replaced with fresh instances. Still, the knight can kill all the dragons in the limit.

is that *when the game is stopped, Bob spends more than Alice since it was him who increasated the ε-bound.*

Therefore, if Alice repeats this procedure, Bob will be the first who violated the global spending limit 1/4. (We assume here that ε is a negative power of 2, so 1/4 is a multiple of ε and there is no last game with reduced budget.) So this approach works for one requirement. This is not directly useful (since we know how to deal with one requirement anyway) but this improvement is important when we combine strategies against different reductions.

Final argument The Alice's strategy in the global game is the priority-type combination of the economical strategies described in the previous section. We assume that Bob's budget in the global game is 1/4, while Alice's budget is 1/2, so she has some reserve of size 1/4. (This increase in Alice's budget is not important in the global game as we have discussed.) This reserve is split into pieces: we consider a computable series $\varepsilon_1, \varepsilon_2, \ldots$ such that $\sum \varepsilon_i < 1/2 - 1/4$; all ε_i are negative povers of 2 (e.g., one may use $1/16, 1/32, 1/64, \ldots$).

To beat the first reduction M_1, Alice uses the economical strategy with parameter ε_1. This means that she analyses the symmetric local game with budget ε_1, finds the point x where the corresponding EXPSPACE-language differs from L and applies the winning strategy in the local game, considering the moves of Bob in the global game as the moves in the local game while it is possible, i.e., while Bob does not exceed the bound ε_1 of the local game. If and when this happens, Alice forgets about this local game, repeats her analysis for the same ε_1 but with current values of k to find new x where the wrong answer can be forced, starts applying the new local strategy until Bob exceeds ε_1-limit, etc.

Alice will do something in parallel against other M_i, but we postpone this discussion. Now we see that the number of restarts is finite, since each restart uses the same ε_1, and Bob cannot spend more than ε_1 infinitely many times. (It does not matter that ε_1 is small as soon as it is not changed during the restart.) And in each game that is abandoned Alice uses less weight than Bob, only in the last game (that is not abandoned; Alice wins in this game) she may use more weight than Bob, and the difference is bounded by ε_1.

How Alice incorporates playing against the next machine M_2 in this scheme? At any moment she may look at the current values of k, perform EXPSPACE-analysis in the ε_2-bounded game against M_2, and start (in parallel) playing this game against M_2-reduction. This game is interrupted if Bob exceeds ε_2 (recall that he does not know anything about ε_i and just plays the global game with budget 1/4), or if M_1-game changes some of the k-values that are used in both games. In the first case the ε_2-analysis is performed again and the game against M_2 is restarted — maybe, with different x and points where the k-values are decreased, but with the same ε_2. If only interrupts of the first kind happen, then the second game always uses ε_2 and therefore is restarted finitely many times, and at each moment Alice spends (in all the games against M_2) not more than Bob, plus ε_2.

However, if an interrupt of the other type happens (due to the decrease in k caused by the game against M_1), then the second game is also restarted, but with ε_3, not ε_2. (Now it is possible that in the abandoned game Alice used more weight than Bob, but only by ε_2). If again the M_2-game is interrupted due to M_1-game, then the new instance of M_2-game uses ε_3, and so on. The priority argument works: since we do not disturb the games against M_1 by M_2-games, then finally the M_1-game stabilizes. After that only the interrupts of the first type are possible for M_2-games, so they all have the same ε_i, and therefore only finitely many interrupts are possible, and Alice wins the global game against M_2.

Some clarification about shared k-values is needed. When performing M_2-analysis, the value of all $k(x)$ involved in M_1-game are considered as fixed. If Alice or Bob change one of them (in M_1-game both players can do this), then the analysis becomes invalid. In fact, it is easier to agree that *any* move in M_1-game (even in the place that is not used in the M_2-game) causes the interrupt of the second type for M_2-game, so it is restarted (with new limit). Still Bob's moves in places that are not used in the M_1-game do not cause the M_2-game to restart, they are just normal moves of Bob in the M_2-game. The restart of the M_1-game also implies the interrupt of the second type for the M_2-game (it is needed since now M_1-game uses other values of k that should be considered as constants for M_2-game). Note that the reverse direction is safe: in M_2-games the values used in the current M_1-game are considered as constants, so M_1-game cannot be disturbed by their change in M_2-game.

In the same way all the other M_i are added sequentially. When in M_i Bob exceeds his budget (and the game is restarted), or if some move is made in the M_i-game, then all following M_j (with $j > i$) get an interrupt of the second type and should be restarted (sequentially, in the order of increasing j). In the first case the M_i-game is restarted also (before all the M_j), using the same bound. All other games use fresh values in the ε_i-sequence for restart.

In this way the total excess of Alice in the global game will never exceed the sum of ε_i, and the global budget for Bob is $1/4$, so Alice never uses more than $1/2$ and wins against all M_i, as required.

Proof of Theorem 13

Clearly the language is in PSPACE, since the number of moves is bounded by the (unary) capitals m_A and m_B.

To show PSPACE-hardness, we reduce from TQBF. (More precisely, we reduce from the complement of TQBF, since our reduction produces GENERAL-GRAPH-GAME instances in which Bob has a winning strategy if and only if the original TQBF instance is a "yes" instance. But PSPACE is closed under complementation so the distinction is immaterial.) The language TQBF consists of true quantified boolean formulae of the form

$$\exists x_1 \forall x_2 \ldots f(x_1, x_2, \ldots, x_n),$$

where f is a boolean formula composed of variables $x_1, x_2, \ldots, x_n$, and the operators $\wedge$, $\vee$ and $\neg$. It suffices (due to the Tseytin transformation) to restrict our attention to instances where f is in conjunctive normal form with 3 literals per clause, i.e. where f takes the form

$$f(x_1, x_2, \ldots, x_n) = C_1 \wedge C_2 \wedge \ldots \wedge C_m,$$

where each clause C_i is of the form

$$C_i = L_{i1} \vee L_{i2} \vee L_{i3},$$

and each literal L_{ik} is either x_j or $\neg x_j$ for some variable j.

Given such a TQBF instance, our reduction creates a GENERAL-GRAPH-GAME instance consisting of $n+1$ games – one game for each variable, and an auxiliary game known as the clause selection game.

The variable game graph G_j for each universally-quantified variable x_j is constructed as follows. Note that the exact move costs will be defined after the graph construction has been outlined, though terms denoting these costs will be assigned to edges in the graph construction.

1. Create vertex labeled x_j marked with a B (i.e. a winning vertex for Bob). The chip for G_j starts on vertex x_j.
2. Create a vertex labeled TA and a vertex labeled FA, each marked with an A (i.e. both are winning positions for Alice). Create directed edges from x_j to both TA and FA, both with cost t.
3. Create a vertex labeled TB, a vertex labeled FB, and a vertex labeled XB, each marked with a B. Create edges (TA, TB) and (FA, FB), both marked with cost t, and edges (TA, XB) and (TB, XB), both with cost $t-1$.
4. Create a vertex labeled XA marked with an A, and an edge (XB, XA), with cost $m_A - jt$.
5. For each literal L_{ik} of clause C_i, if $L_{ik} = x_j$ or $L_{ik} = \neg x_j$, create vertices labeled C_iTA and C_iFA marked with A and vertices labeled C_iTB and C_iFB marked with B. Add edges (TB, C_iTA) and (FB, C_iFA) with costs both set to l_{ik}.
   Add edges (C_iTA, C_iTB) and (C_iTA, C_iTB). If $L_{ik} = x_j$, then set (C_iTA, C_iTB)'s cost to $l_{ik}-1$ and (C_iTA, C_iTB)'s cost to l_{ik}. Otherwise set (C_iTA, C_iTB)'s cost to l_{ik} and (C_iTA, C_iTB)'s cost to $l_{ik}-1$.

For each existentially-quantified variable, the corresponding variable game's structure is mostly identical to that of universally-quantified variables. The adjustments we need to make are entirely to steps 2 and 3 of the preceeding construction.

2'. Create a vertex labeled x_jA marked with an A. Create a directed edge from x_j to x_jA, with cost t.
3'. Create a vertex labeled TB, a vertex labeled FB, and a vertex labeled XB, each marked with a B. Create edges (x_jA, TB) and (x_jA, FB), both marked with cost t, and an edge (x_jA, XB) with cost $t-1$.

Finally, we define the clause selection game:

1. Create a vertex labeled C, marked with B.
2. For each clause C_i, create a vertex labeled C_iA marked with A, and a vertex labeled C_iB marked with B. Create an edge (C, C_iA) with edge cost c_i (again to be determined below), and an edge (C_iA, C_iB) also with edge cost c_i.

An example of the game graphs produced by applying the reduction to a TQBF instance is supplied in the Appendix.

Intuitively, each variable game consists of a *true subgame*, a *false subgame*, and an *override subgame*. (The latter being the subgame consisting of the XA and XB vertices.) Alice always chooses which game to progress next – Bob must always respond in the game Alice last moved in, since his victory criterion has him losing if he is losing in any one game. In optimal play, Alice should take exactly $n+4$ moves, and her first n moves should progress each variable game into either its true subgame or its false subgame, in order corresponding to that which the corresponding variables were quantified in the original TQBF instance. In games corresponding to universally-quantified variables, Alice's move itself determines which subgame is entered into; in those corresponding to existentially-quantified variables, Bob's response determines the subgame entered into. The override subgames exist to allow Bob to punish Alice for progressing a variable game out-of-order. After each player has made n moves, the overall game state corresponds to a truth assignment for the variables occurring in the formula f, and Alice should have a winning strategy if and only if some clause C_i is left unsatisfied by said truth assignment.

We define the move costs and starting capitals as follows. Set $b = 3(n+m)+10$, and set

$$\begin{aligned}
t &= b^{12} \\
l_{ik} &= b^{12} + 2b^{10} + ib^{2k} \text{ for each } k \in \{1,2,3\} \\
c_i &= b^{12} + 3b^{10} + b^8 - \sum_{k=1}^{3} ib^{2k} \\
m_A &= (n+4)b^{12} + 9b^{10} + b^8 \\
m_B &= m_A - 1
\end{aligned}$$

Note that all of these values are polynomial in m and n, so the GENERAL-GRAPH-GAME instance outputted by the reduction has size polynomial in the size of the input TQBF instance even if these costs are encoded in unary.

The motivation for the terms appearing in the costs and starting capitals is as follows. The terms of order b^{12} ensure that each player is only allowed to make $n+4$ moves. The terms of order b^{10} ensure that of Alice's $n+4$ moves, only one move with a cost of form c_i, and only three moves with costs of the form l_{ik}, are made. (The other n moves made by Alice must cost exactly t.) The terms of orders b^2 through b^8 are crafted such that Alice's only winning strategy, after the first n moves are made, consisting of making moves which correspond to scrutinizing a single clause left unsatisfied by the truth assignment generated by the first $2n$ moves of play, is as follows:

Alice's clause C_i strategy: let $C_i = L_{i1} \vee L_{i2} \vee L_{i3}$. For each $k \in \{1, 2, 3\}$, if $L_{ik} = x_j$ or $\neg x_j$, move to $C_i?A$ in the variable game for x_j. Also move to C_iA in the clause selection game.

($C_i?A$ denotes either C_iTA or C_iFA, only one of which is reachable in a given variable game after the first $2n$ moves have passed in optimal play.)

Finally, note that Bob's starting capital differs from Alice's starting capital by 1, and that the moves available to Bob in response to any given move by Alice either match the cost of Alice's move, or are discounted by 1. Said discounts are available to Bob either in situations where Alice had made a move corresponding to scrutinizing a satisfied literal, or when Bob chooses to progress a variable game to its override subgame. If Bob can ever make one of these discount moves, and assuming such a move doesn't allow Alice access to an affordable response in an override subgame, then Bob subsequently has an easy winning strategy, since from that point onwards his capital matches Alice's. On the other hand, if Alice can make $n + 4$ moves which exhaust her starting capital (only possible via Alice's clause C_i strategy stated above) and never allow Bob a safe discount, then Alice wins.

Example for reduction in Theorem 13 Consider the TQBF instance of form

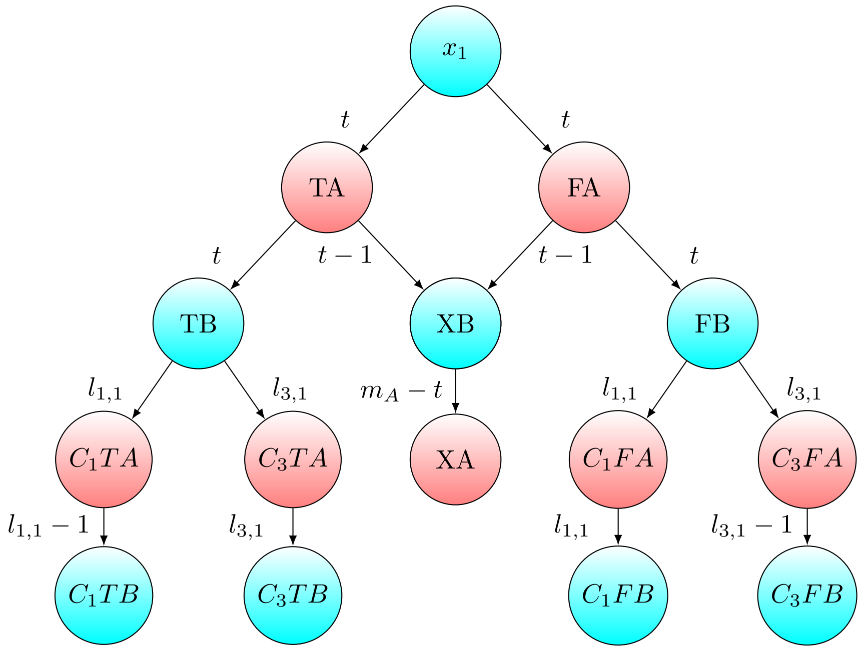
$$\forall x_1 \exists x_2 \forall x_3 \exists x_4 (C_1 \wedge C_2 \wedge C_3),$$

with clauses as follows:

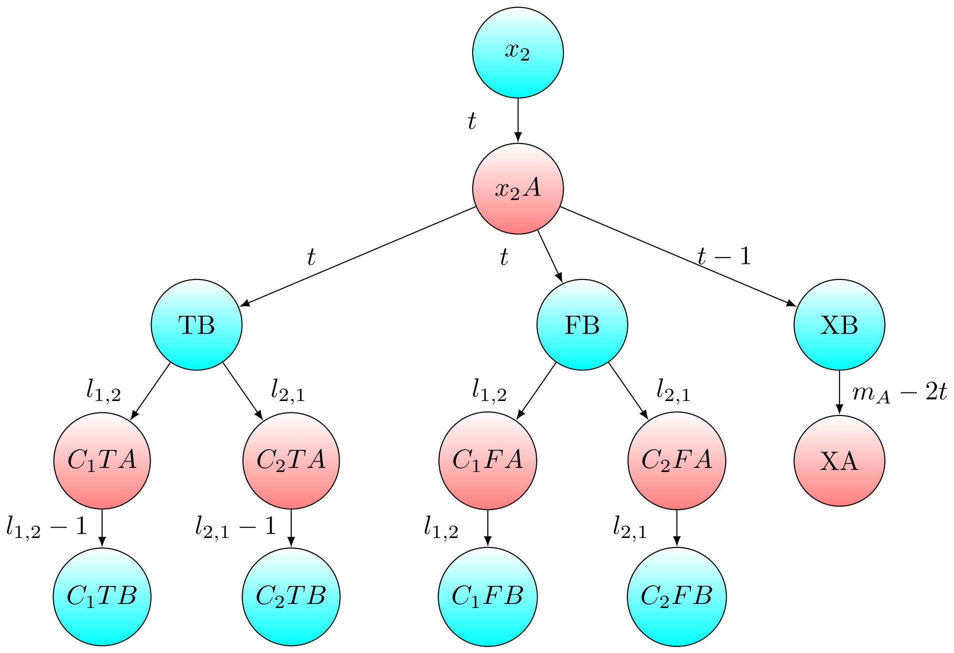
$$\begin{aligned} C_1 &= x_1 \vee x_2 \vee \neg x_3 \\ C_2 &= x_2 \vee x_3 \vee \neg x_4 \\ C_3 &= \neg x_1 \vee \neg x_3 \vee x_4 \end{aligned}$$

The reduction outlined in Theorem 4 produces the games illustrated in the following diagrams.

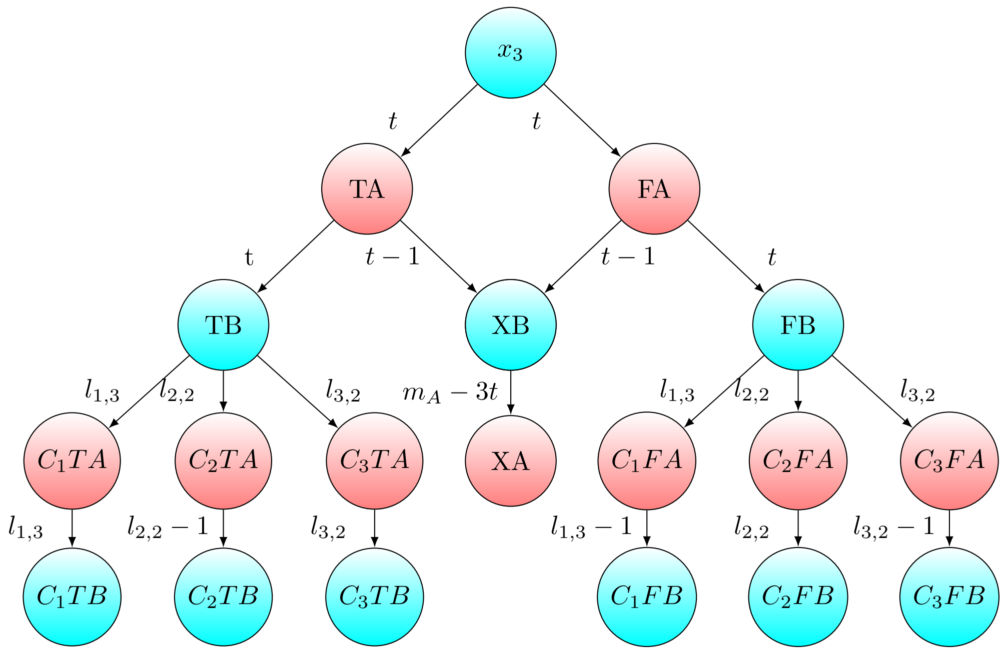
Variable game for x_1:


x_1
t
t
TA
FA
t
$t-1$
$t-1$
t
TB
XB
FB
$l_{1,1}$
$l_{3,1}$
m_A-t
$l_{1,1}$
$l_{3,1}$
C_1TA
C_3TA
XA
C_1FA
C_3FA
$l_{1,1}-1$
$l_{3,1}$
$l_{1,1}$
$l_{3,1}-1$
C_1TB
C_3TB
C_1FB
C_3FB

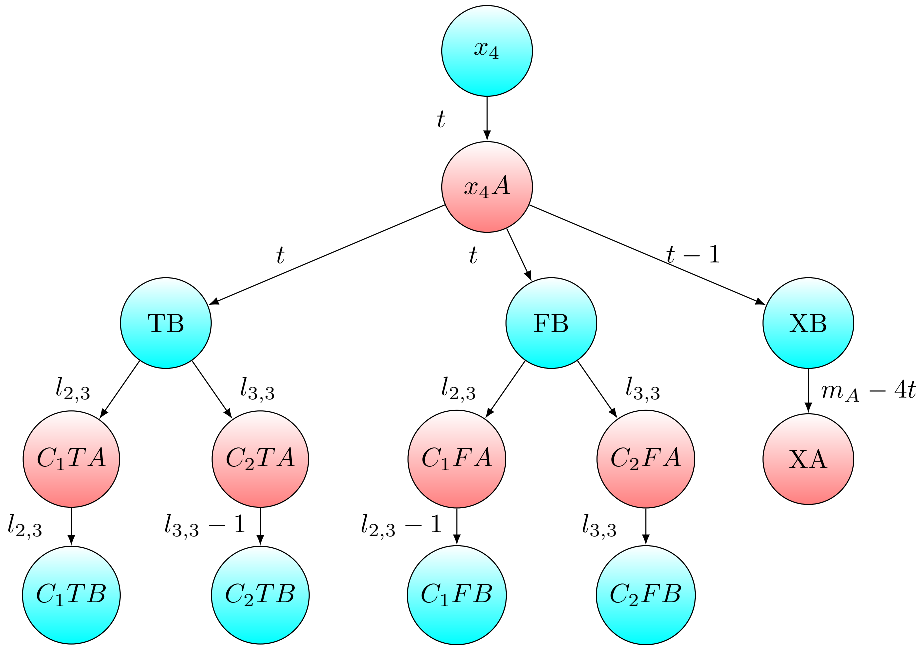
Variable game for x_2:



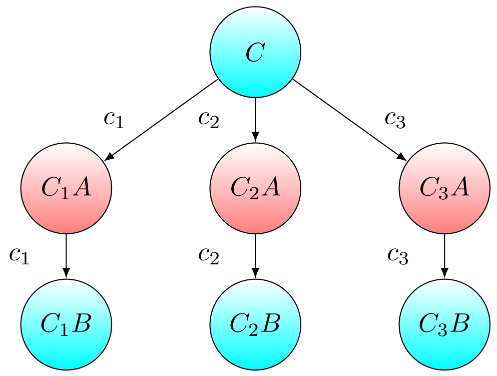
Variable game for x_3:



Variable game for x_4:



Clause selection game:



Completion of the proof of Theorem 4

The correctness of the reduction follows from a succession of claims, established below. For succinctness, the statements of these results assume optimal play by both Alice and Bob.

Alice cannot make more than $n + 4$ moves. Clear from the fact that each move costs $\geq b^{12}$ dollars, and $(n+5)b^{12}$ exceeds Alice's starting capital.

Alice must make exactly $n+4$ moves. Crucially for this claim, there are play sequences involving Alice winning in fewer than $n+4$ moves involving her winning a variable game's override subgame (i.e. moving into a position labelled XA) – however, in optimal play, we can assume this never happens, as Bob can always choose to avoid progressing a variable's game into its override subgame if Alice has sufficient remaining capital to respond in that subgame. Having ruled out this possibility, if Alice makes fewer than $n+4$ moves, then the total cost of Bob's response moves can't be larger than $(n+4)b^{12}$, which is less than Bob's starting capital.

Alice must move at least once on each variable game. In particular, exactly n of the aforementioned $n+4$ moves which Alice must make, must consist of making an initial move on a variable game, thereby progressing it to either its true or false subgame. This follows from the fact that if fewer than n of Alice's moves consist of initial moves in variable games, then she must make at least 5 moves with costs of forms either c_i or l_{ik}, so the total cost of Alice's $n+4$ moves is $\geq (n+4)b^{12}+10b^{10}$, which exceeds Alice's starting capital.

Alice's j-th move must be the initial move on the x_j variable game. We establish this by induction on j. Assuming true for $0,\ldots,j-1$, by the time of Alice's j-th move, no moves can have been made in the variable game for x_j, as all moves up until this point have happened in the games for variables $x_1,\ldots,x_{j-1}$. If Alice's j-th move is *not* the initial move for variable game x_j, then by the previous claim, Alice must eventually move in the game for x_j, so assume that move is made on Alice's j'-th turn, with $j<j'$. Then for Bob's j'-th move, Bob has a response available to him of moving to position XB (i.e. progressing to the override subgame) in the x_j game, obtaining for him a discount of 1, and subsequently Alice's remaining capital is $< m_A - jt$, so she cannot progress the subgame to the XA position. This gives Bob an easy winning strategy, as, having obtained a discount of 1 on a response move, he is now able to afford to match each of Alice's subsequent moves with a move of equal cost.

Thus far we have established that, in optimal play, the first $2n$ moves will consist of progressing each of the variable games into either its true subgame or its false subgame, in an order which corresponds to the order of quantification of the variables in the original TQBF instance. Thus, in optimal play, the first $2n$ moves will generate a game state where a subgame has been entered into on each variable game, which corresponds to a truth assignment in which Alice has determined the universally-quantified variables and Bob has determined the existentially-quantified variables, and the determination of each variable x_j has been made with knowledge of the determinations of variables $x_1,\ldots,x_{j-1}$.

It remains at this point to establish the following claim:

Alice has a winning strategy if and only if the induced truth assignment leaves some clause C_i unsatisfied. For the "if" direction of this claim, Alice can win with her clause C_i strategy, which we repeat here:

> **Alice's clause C_i strategy:** let $C_i = L_{i1} \vee L_{i2} \vee L_{i3}$. For each $k \in \{1, 2, 3\}$, if $L_{ik} = x_j$ or $\neg x_j$, move to $C_i?A$ in the variable game for x_j. Also move to C_iA in the clause selection game.

Note that the order in which Alice makes her moves at this point is irrelevant, as Bob's responses are forced and the total capital Bob will require to respond to Alice's moves is unchanged by the order of Alice's moves.

Implementing the strategy above costs Alice $l_{i1}+l_{i2}+l_{i3}+c_i = 4b^{12}+9b^{10}+b^8$ capital to implement, which is Alice's entire remaining capital. Assuming C_i is unsatisfied by the induced truth assignment, then by construction Bob's response also costs $l_{i1} + l_{i2} + l_{i3} + c_i$, which exceeds Bob's remaining capital by exactly 1.

The above strategy fails if clause C_i is satisfied by the induced truth assignment, as by construction Bob gets a discount on at least one of his variable game responses. To establish the "only if" direction of the above claim, it suffices to establish that Alice's clause C_i strategy, for some clause C_i, is the only possible winning strategy for Alice. We establish this via the following succession of claims.

Alice must select a move from the clause selection game. If she doesn't, then she selects 4 moves from variable games, with total cost $\leq 4b^{12}+8b^{10}+4b^7$, against which Bob can easily afford the counterplay.

From this stage onward, we disregard the b^{12}, b^{10}, and b^8 terms in the remaining capitals and move costs chosen, as they will always sum to $4b^{12} + 9b^{10} + b^8$.

Since Alice must choose exactly one move in the clause selection game, assume that move is to C_iA. Then Alice has $\sum_{k=1}^{3} ib^{2k}$ capital remaining, Bob has 1 less than this amount remaining, and Alice has (assuming w.l.o.g. that she makes the move in the clause selection game first) 3 moves remaining.

Alice must select at least one move costing l_{j3} for some clause C_j. If she doesn't, then her moves cost $\leq 3b^5$, and Bob has more than enough capital for counterplay.

Said move costing l_{j3} must be the move costing l_{i3}. It can't be a move costing l_{j3} for $j > i$, otherwise this move alone costs $\geq (i+1)b^6$ which is greater than Alice's remaining budget. If it's l_{j3} for $j < i$, then the $l_{(i-j)3}$ cost move must also be chosen by Alice to exhaust the b^6-order term in Bob's remaining budget. But that leaves Alice with one remaining move in which to exhaust both the b^4-order term and the b^2-order term in Bob's remaining budget, which is impossible by construction, so Bob wins handily.

At this point, analogous arguments establish that Alice must select the moves costing l_{i2} and l_{i1}, i.e. she must implement exactly her clause C_i strategy, as required.

Proof of Theorem 12

First note that this language belongs to PSPACE because the number of moves in the game is bounded by poly(n).

For proving PSPACE-hardness we first note the following. As in games before we can assume that a player miss his or her move if the current value of $M^O(x)$ suits for him or for her. Therefore we can assume the following. A losing player adds strings $a_1, a_2, \ldots, a_k$ until $M^O(x)$ does not change. Then the second player adds $b_1, b_2, \ldots, b_k$ until the value $M^O(x)$ does not change and so on. The set of strings whose addition change the value of $M^O(x)$ is called *a big move.* Let A, B be big moves and $A \subseteq B$. Then big move A is not worse than big move B (if a player has a winning strategy with move B then he or she has a winning strategy with move A).

We claim that language TQBF reduces to tt-GAME. The language TQBF contains the following formulas:

$$\exists x_1 \forall y_1 \ldots \exists x_n \forall y_n \exists x_{n+1} f(x_1, y_1, \ldots, x_n, y_n, x_{n+1}). \tag{2}$$

(By technical reasons we consider formulas where $\exists$ one more than $\forall$. It is clear that this language is still PSPACE-complete.)

Construct a function v and a machine with access to oracle O that inputs x—formulas of kind (2). This formula is true iff $(x, n+1, (n+1)2^{n+1}) \in$ tt-GAME.

This machine on input (2) makes requests to strings x_1, $\neg x_1$, y_1, $\neg y_1 \ldots$, y_n, $\neg y_n$, x_{n+1}, $\neg x_{n+1}$, and also to additional strings r and a.

The function v defines by the following way: $v(x_i) = v(\neg x_i) = 1$,$v(y_i) = v(\neg y_i) = 2^{n+1}$ for every i, $v(r) = 2^{n+1}$, $v(a) = 1$.

After these requests the machine works by the following way:

- if $r \notin O$ then the machine outputs $g(x_1, \neg x_1, y_1, \neg y_1 \ldots, x_n, \neg x_n, y_n, \neg y_n, x_{n+1}, \neg x_{n+1})$, we describe function g later.
- if $r \in O$ and $a \in O$ then the machine outputs 1.
- if $r \in O$ and $a \notin O$ then machine outputs the following. If oracle O output *correct* answers for every pair x_i and $\neg x_i$ (this is mean that answers are opposite) and for every pair y_i and $\neg y_i$ then the machine outputs f at the corresponding valuation of x_i and y_i.
  If the oracle output non-correct answers then the machine outputs 1.

Now we describe function g: $g := h(x'_1, \ldots x'_{n+1}, y'_1, \ldots y'_n)$, where $x'_i = x_i \vee \neg x_i$ and $y'_i = y_i \vee \neg y_i$. Finally, describe function h:

- if $y'_i = 0$ for every i, then h equals x'_1.

– else h is equal x'_{i+1}, where i is maximal number such that $y'_i = 1$.

Now we show that this reduction is correct. For this we describe class *adequate* strategies for Alice and Bob. An adequate strategy is the following: Alice takes x_1 or $\neg x_1$, Bob takes y_1 or $\neg y_1$ and so on. Alice takes x_{n+1} or $\neg x_{n+1}$, Bob takes r and game over (the players ran out of money). A win defines by $f(x_1, y_1, \dots, x_n, y_n, x_{n+1})$.

To finish the proof it is enough to show that if a player has a winning strategy then he or she has an adequate winning strategy.

First prove it for Alice. Indeed, if players made i adequate pair of moves then the current value of $M^O(x)$ is equal to 0. So, to change the current value of $M^O(x)$ Alice can only take x_{i+1} or $\neg x_{i+1}$ (therefore, Alice will act according an adequate strategy).

Now we prove that Bob also should not deviate from an adequate strategy. Let players made adequate moves and Alice took x_i or $\neg x_i$ by her last move. According to an adequate strategy Bob must take y_i or $\neg y_i$, or r if $i = n + 1$.

If $i = n+1$ then g is equal to 1 then the last chance of Bob is to take r (after this game over because players ran out of money).

If $i \leq n$ then change the current value of $M^O(x)$ has to take y_j or $\neg y_j$, where $j \geq i$. We need to show that if Bob take $j > i$ (and does not take y_j or $\neg y_j$) then Bob definitely looses. Indeed, Alice can take x_{j+1}, x_{j+2}, $\dots x_{n+1}$. After this function g is equal to 1 and Alice has at least one dollar. Since g is equal to 1 Bob has to take r but then Alice can take a and win.

Proof of Theorem 12

[6] First show that language T-GAME belongs to EXPSPACE. Indeed, a machine can require strings of polynomial length. So, there is no sense to add strings with higher length to an oracle. The number of such strings is exponent hence the number of moves is bounded by exponent, So the game belongs to EXPSPACE.

To prove EXPSPACE-hardness consider language SUCCINCT-3-SAT-TQBF. This language contains some circuits that code exponential size 3-CNF by the following way: an input for a circuit are three literals, a circuit outputs 1 if the 3-CNF has a disjunct consisting from these three literals and outputs 0 otherwise. Let some circuit codes 3-CNF $f(x_1, \dots, x_n)$. This circuit belongs to language SUCCINCT-3-SAT-TQBF if the formula $\exists x_1 \forall x_2 \dots f(x_1, \dots, x_n)$ is true.

We affirm that language SUCCINCT-3-SAT-TQBF is EXPSPACE-hard by a standard argument.

Before constructing a reducibility we make some notes about T-GAME.

This is game on a marked tree—see Figute 4 Every path at this tree corresponds to a calculation by a machine with some oracle O. Initially (when O is empty) the path is the most left. During the game the path shifts to the right.

The notion about big moves for tt-GAME in proof of Theorem 11 holds also for T-GAME. Here it means the following. We can assume that a player making a

[6] In the proof we use ideas from [9]

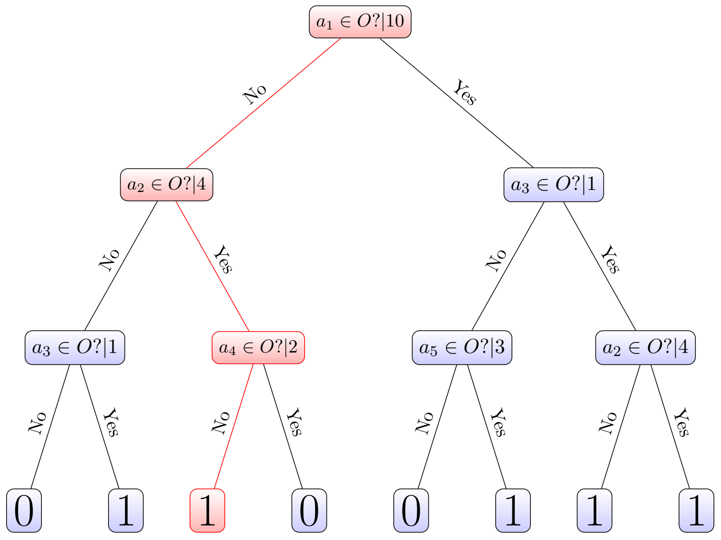


Fig. 4. The values of function v are at the right parts of vertices. The current path is imaged by the red line

move had a losing position before this move. This move must change the current path (In Figure 4 Bob must make a move because the current value is 1. He can take a_1 or a_4.)

Now we describe reducibility.

Consider TQBF with variables $x_1, y_1, \dots x_{2^n}, y_{2^n}, x_{2^n+1}$ (we can assume that TQBF has such number of variables this is not important for EXPSPACE-hardness).

We construct a marked tree T; at the game at T Alice wins if the TQBF is true and Bob wins if the TQBF is false.

The players' capitals: Alice has $2^n + 1$ (by default she spends it on $2^n + 1$ vertices x_i or $\neg x_i$ that cost 1. Bob has $2^{n+1}(2^n + 1) + 2^n$ (by default he spends them on 2^n vertices y_i or $\neg y_i$ that cost 2^{n+1} + on vertex r that costs 2^{n+1} + some poly(n) as "pocket money").

First we describe a general plan of tree T. Instead of $x \in O?$ we just write x. The general plan is shown in the Figure 5. The idea is similar to the proof of Theorem 11. In particular, the sense of vertices r and a the same is in that proof. The sense of G is the same as the sense of function g. In the proof of Theorem 11 function g is expressed by function h; here tree G is some superstructute on tree H—see Figure 6. The sense of this construction is the following. Players by default make moves x'_1, y'_1, ... Alice can not make other moves to change the current move. If Bob takes something wrong then Alice can win in this sub-tree and does not spend all her money. Then she can take a and win in the game.

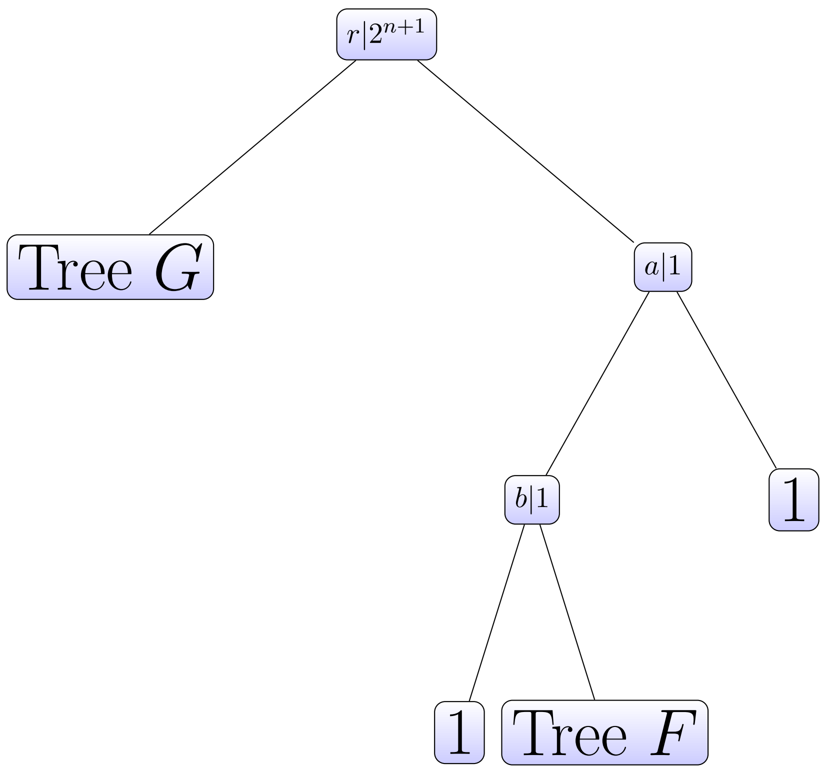


Fig. 5. General plan

The tree G is similar to the tree H, however instead of asks about x'_i (or y'_i) it asks about x_i and $\neg x_i$ and then take OR from the answers (see Figure 7) It remains to describe tree F. It is a tree whose vertices at upper levels are some fresh strings with cost 1. The paths of this tree codes all kinds of disjuncts with tree literals. After this machine's action depends on the input (recall that the input is a succinct 3-CNF). The machine verifies existence of the corresponding disjunct in the 3-CNF. If 3-CNF does not contain the disjunct then the machine outputs 1. If 3-CNF contains the disjunct then machine asks oracle: does it has at least one of corresponding literals. If the answer is "yes" then the machine outputs 1; otherwise it outputs 0.

The description of the tree is complete. It is clear that this tree can be modeled as a calculation of some polynomial-time Turing machine with oracle. So, we construct a reducibility. Show that this reducibility is correct.

Describe the class of adequate strategies for the players. At first they are similar to adequate strategies from the proof of Theorem 11: Alice takes x_i or $\neg x_i$, Bob takes y_i or $\neg y_i$. Then Bob takes r. If the current formula is false (i.e. it contains false disjunct) then Bob takes b and then in tree F he chooses a path

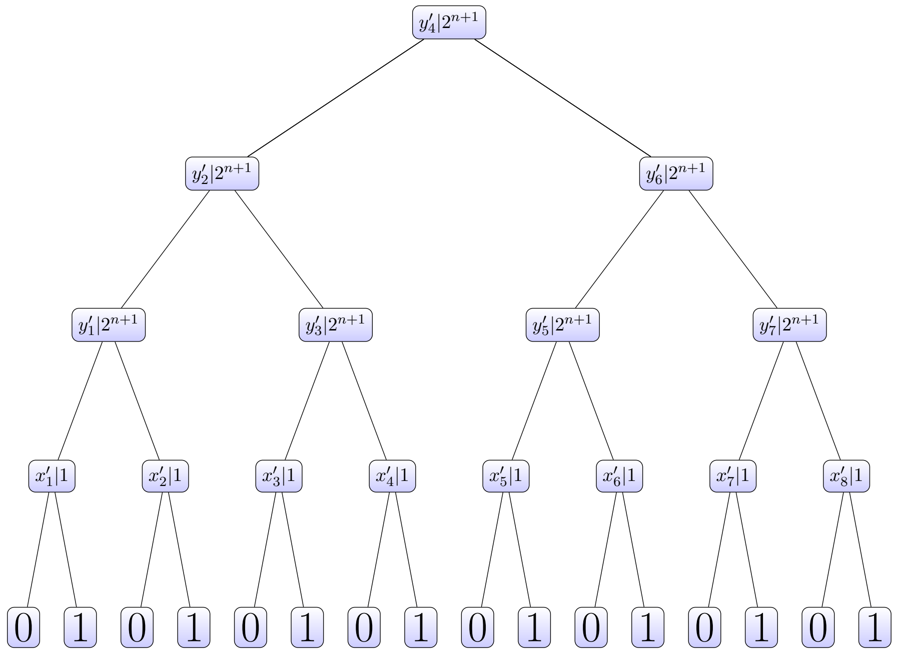


Fig. 6. Tree H.

corresponding to the false disjunct in the formula. Here Bob spend his "pocket money"—since the depth of the tree is some polynomial he has enough money for it.

It remains to show that players should use adequate strategies. For Alice it is clear—she has not another choice (if Bob also use an adequate strategy). For Bob we have a similar reasoning as in the proof of Theorem 11. If in tree G Bob takes wrong vertex then Alice can win in G and then take a. Bob has not sense to take Alice's vertices in G because it can only mess up his situation.

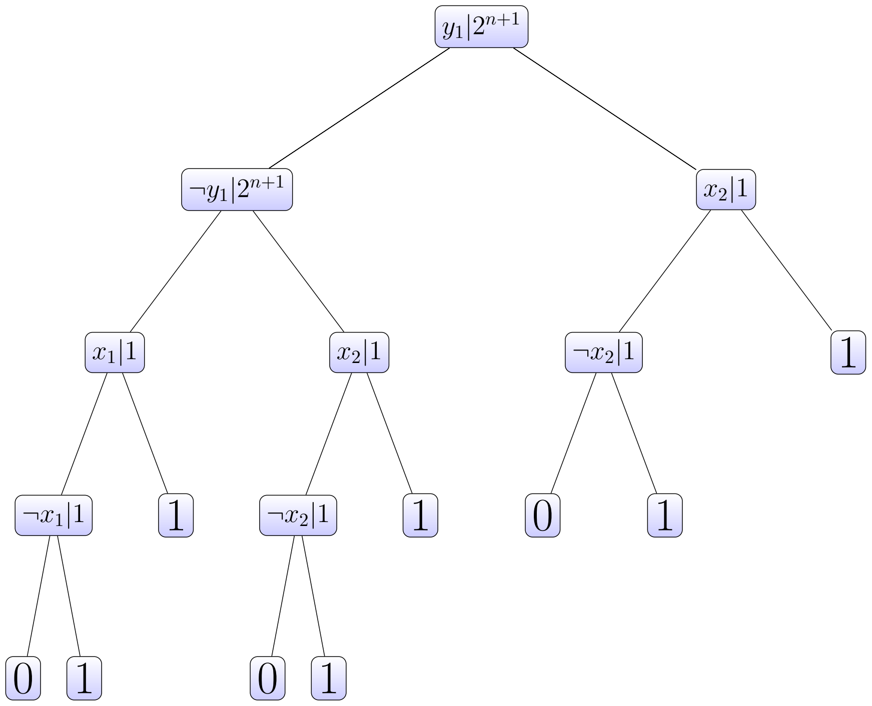


Fig. 7. Tree G: the transformation of the left quarter of tree H.