Playing Games with Boxes and Diamonds* **

Rajeev Alur[1], Salvatore La Torre[2], and P. Madhusudan[1]

[1] University of Pennsylvania
[2] Università degli Studi di Salerno

Abstract. Deciding infinite two-player games on finite graphs with the winning condition specified by a linear temporal logic (LTL) formula, is known to be 2EXPTIME-complete. The previously known hardness proofs encode Turing machine computations using the *next* and/or *until* operators. Furthermore, in the case of model checking, disallowing next and until, and retaining only the *always* and *eventually* operators, lowers the complexity from PSPACE to NP. Whether such a reduction in complexity is possible for deciding games has been an open problem. In this paper, we provide a negative answer to this question. We introduce new techniques for encoding Turing machine computations using games, and show that deciding games for the LTL fragment with only the *always* and *eventually* operators is 2EXPTIME-hard. We also prove that if in this fragment we do not allow the *eventually* operator in the scope of the *always* operator and vice-versa, deciding games is EXPSPACE-hard, matching the previously known upper bound. On the positive side, we show that if the winning condition is a Boolean combination of formulas of the form "eventually p" and "infinitely often p," for a state-formula p, then the game can be decided in PSPACE, and also establish a matching lower bound. Such conditions include safety and reachability specifications on game graphs augmented with fairness conditions for the two players.

1 Introduction

Linear temporal logic (LTL) is a specification language for writing correctness requirements of reactive systems [13, 11], and is used by verification tools such as SPIN [8]. The most studied decision problem concerning LTL is *model checking*: given a finite-state abstraction G of a reactive system and an LTL formula φ, do all infinite computations of G satisfy φ? The corresponding *synthesis* question is: given a game graph G whose states are partitioned into system states and environment states, and an LTL formula φ, consider the infinite game in which the protagonist chooses the successor in all system states and the adversary chooses the successor in all environment states; then, does the the protagonist have a strategy to ensure that all the resulting computations satisfy φ? Such a game-based interpretation for LTL is useful in many contexts: for synthesizing controllers from specifications [14], for formalizing compositionality requirements

* Detailed proofs are available at `http://www.cis.upenn.edu/~madhusud/`

** Supported in part by ARO URI award DAAD19-01-1-0473, NSF awards CCR9970925 and ITR/SY 0121431. The second author was also supported by the MIUR grant project "Metodi Formali per la Sicurezza e il Tempo" (MEFISTO) and MIUR grant ex-60% 2002.

such as *realizability* [1] and *receptiveness* [6], for specification and verification of open systems [3], for modular verification [10], and for construction of the most-general environments for automating assume-guarantee reasoning [2]. In the contexts of open systems and modular verification, this game is played in the setting where a module is considered as the protagonist player, and its environment, which may consist of other concurrent modules in the system that interact with this module, is taken as the adversary.

An LTL formula is built from state predicates (Π), Boolean connectives, and temporal operators such as *next*, *eventually*, *always*, and *until*. While the model checking problem for the full LTL is known to be PSPACE-complete, the fragment $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ that allows only *eventually* and *always* operators (but no *next* or *until*), has a small model property with NP-complete model checking problem [15]. Deciding games for the full LTL is known to be 2EXPTIME-complete [14]. The hardness proof, like many lower bound proofs for LTL, employs the until/next operators in a critical way to relate successive configurations. This raises the hope that deciding games for $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ has a lower complexity than the full LTL. In this paper, we provide a negative answer to this question by proving a 2EXPTIME lower bound.

The proof of 2EXPTIME-hardness is by reduction of the halting problem for alternating exponential-space Turing machines to deciding games with winning condition specified by formulas that use only the always and eventually operators. The reduction introduces some new techniques for counting and encoding configurations in game graphs and formulas. We believe that these techniques are of independent interest. Using these techniques we show another hardness result: deciding games for the fragment $\mathcal{B}(L_{\Diamond,\wedge,\vee}(\Pi))$ is EXPSPACE-hard. This fragment contains top-level Boolean combinations of formulas from $L_{\Diamond,\wedge,\vee}(\Pi)$, the logic of formulas built from state predicates, conjunctions, disjunctions, and eventually operators. $\mathcal{B}(L_{\Diamond,\wedge,\vee}(\Pi))$ is known to be in EXPSPACE [4], while $L_{\Diamond,\wedge,\vee}(\Pi)$ is known to be in PSPACE [12], so our result closes this complexity gap.

Finally, we consider the fragment $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ that contains Boolean combinations of formulas of the form $\Box\Diamond\, p$, where p is a state predicate. Complexity for formulas of specific form in this class is well-known: generalized Büchi games (formulas of the form $\wedge_i \Box\Diamond p_i$) are solvable in polynomial time, and Streett games ($\wedge_i(\Box\Diamond p_i \rightarrow \Box\Diamond q_i)$) are CO-NP-complete (the dual, Rabin games are NP-complete) [7]. We show that the Zielonka-tree representation of the winning sets of vertices [17] can be exploited to get a PSPACE-procedure to solve the games for the fragment $\mathcal{B}(L_{\Box\Diamond}(\Pi))$. This logic is of relevance in modeling *fairness* assumptions about components of a reactive system. A typical fairness requirement such as "if a choice is enabled infinitely often then it must be taken infinitely often," corresponds to a Streett condition [11]. Such conditions are common in the context of concurrent systems where it is used to capture the fairness in the scheduling of processes. In games with fairness constraints, the winning condition is modified to "if the adversary satisfies all the fairness constraints then the protagonist satisfies its fairness constraints and meets the specification" [3]. Thus, adding fairness changes the winning conditions for specifications from a logic $\mathcal{L}$ to Boolean combinations of $L_{\Box\Diamond}(\Pi)$ and $\mathcal{L}$ formulas. We show that the PSPACE upper bound holds for fair games for specifications in $\mathcal{B}(L_{\Diamond,\wedge}(\Pi))$ containing Boolean combinations of formulas that are built from state predicates, conjunctions, and eventually operators, and can specify combinations of invariant and termination properties. This result has been used to show that the model checking of the game-based temporal logic Alternating Temporal Logic is in PSPACE under the strong fairness requirements [3]. We conclude by showing that deciding games for formulas of the form "Streett implies Streett" (that is, $\wedge_i(\Box\Diamond p_i \to \Box\Diamond q_i) \to \wedge_j(\Box\Diamond r_j \to \Box\Diamond s_j)$) is PSPACE-hard.

2 LTL fragments and game graphs

2.1 Linear Temporal Logic

We first recall the syntax and the semantics of linear temporal logic. Let $\mathcal{P}$ be a set of propositions. Then the set of *state predicates* Π is the set of boolean formulas over $\mathcal{P}$. We define temporal logics by assuming that the atomic formulas are state predicates. A *linear temporal logic* (LTL) formula is composed of state predicates (Π), the Boolean connectives *negation* ($\neg$), *conjunction* ($\wedge$) and *disjunction* ($\vee$), the temporal operators *next* ($\bigcirc$), *eventually* ($\Diamond$), *always* ($\Box$), and *until* ($\mathcal{U}$). Formulas are built up in the usual way from these operators and connectives, according to the grammar

$$\varphi := p \,|\, \neg\varphi \,|\, \varphi \wedge \varphi \,|\, \varphi \vee \varphi \,|\, \bigcirc\varphi \,|\, \Diamond\, \varphi \,|\, \Box\, \varphi \,|\, \varphi\, \mathcal{U}\, \varphi,$$

where $p \in \Pi$. LTL formulas are interpreted on ω-words over $2^{\mathcal{P}}$ in the standard way [13].

In the rest of the paper we consider some fragments of LTL. For a set of LTL formulas Γ, we denote by $L_{op_1,\ldots,op_k}(\Gamma)$ the logic built from the formulas in Γ by using only the operators in the list $op_1, \ldots, op_k$. When the list of operators contains only the Boolean connectives we use the notation $\mathcal{B}(\Gamma)$, i.e., $\mathcal{B}(\Gamma) = L_{\neg,\wedge}(\Gamma)$. In particular, $\Pi = \mathcal{B}(\mathcal{P})$. As an example, the logic $L_{\Diamond,\wedge}(\Pi)$ is the one defined by the grammar $\varphi := p \,|\, \varphi \wedge \varphi \,|\, \Diamond\, \varphi$, where $p \in \Pi$, and the logic $\mathcal{B}(L_{\Diamond,\wedge}(\Pi))$ contains Boolean combinations of the formulas from $L_{\Diamond,\wedge}(\Pi)$.

2.2 LTL games

A *game graph* is a tuple $G = (\Sigma, V, V_0, V_1, \gamma, \mu)$ where Σ is a finite set of labels, V is a finite set of vertices, V_0 and V_1 define a partition of V, $\gamma : V \to 2^V$ is a function giving for each vertex $u \in V$ the set of its *successors* in G, and $\mu : V \to \Sigma$ is a labeling function. For $i = 0, 1$, the vertices in V_i are those from which only player i can move and the allowed moves are given by γ. A *play* starting at x_0 is a sequence $x_0 x_1 \ldots$ in V^* or V^ω such that $x_j \in \gamma(x_{j-1})$, for every j. A *strategy* for player i is a total function $f : V^* V_i \to V$ mapping each finite play ending in V_i into V (it gives the moves of player i in any play ending in V_i). A play $x_0 x_1 \ldots$ is consistent with f if for all $x_j \in V_i$ with $j \geq 0$, $f(x_0 \ldots x_j) = x_{j+1}$.

In this paper, we focus on determining the existence of strategies for player 0. For this reason, player 0 is called the *protagonist*, while player 1 is the *adversary*. Unless specified otherwise, by 'strategy' we mean a strategy for the protagonist. Moreover, we consider game graphs along with winning conditions expressed by LTL formulas (LTL *games*). Formally, an LTL game is a triple (G, φ, u), where G

is a game graph with vertices labeled with subsets of atomic propositions, φ is an LTL formula and u is a vertex of G. A strategy f in (G, φ, u) is winning if all infinite plays consistent with f and starting at u satisfy φ. The decision problem for an LTL game (G, φ, u) is to determine if there exists a winning strategy for the protagonist.

3 Lower bound results

When proving lower bounds for LTL games, the usual technique is to code the acceptance problem for alternating Turing machines [16, 12]. The crux in such a proof is to detect, using the LTL specification, that the content of the i^{th} cell in a configuration is in accordance with the $(i-1)^{th}$, i^{th} and $(i+1)^{th}$ cells of the previous configuration. In a reduction from 2EXPTIME (i.e. alternating EXPSPACE), typically, the cell numbers are explicitly encoded using a sub-word of bits in the configuration sequence; these numbers can be read by zooming into the i^{th} cell in some configuration using the $\Diamond$ operator, using the $\bigcirc$ operator to read the cell numbers and using $\mathcal{U}$ operator to access the next configuration. Here the $\mathcal{U}$ operator can be used instead of $\bigcirc$, but the $\mathcal{U}$ cannot be replaced by $\Diamond$. In an EXPSPACE reduction (i.e. alternating EXPTIME), the configuration numbers are also encoded explicitly, and one can just use the $\Diamond$ and $\bigcirc$ operators to access three cells of a configuration and the appropriate position in the next configuration. Hence both proofs use the $\bigcirc$ operator and the 2EXPTIME reduction uses the $\mathcal{U}$ operator crucially.

The primary difficulty in the lower bounds we present is in dealing with reductions in the absence of the $\bigcirc$ and $\mathcal{U}$ operators. The $\Diamond$ operator can basically check only for subsequences and can hence "jump" arbitrarily far making it difficult to read the cell and configuration numbers. The main idea to solve this, which we call the *matching trick* below, is to introduce a "sandwiched" encoding of addresses which forces the reading of cell numbers to be contiguous. This then yields an EXPSPACE lower bound for $\mathcal{B}(L_{\Diamond,\wedge,\vee}(\Pi))$.

Then we consider $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ where one can nest $\Diamond$ and $\Box$ operators. Though this does not allow us to check the required property on an entire sequence of configurations, it allows us to check it for the *last two* configurations in a sequence. By giving the adversary the ability to stop the sequence of configurations at any point, we can ensure that the entire sequence satisfies the property, which leads to a 2EXPTIME lower bound for this fragment.

3.1 Alternating Turing Machines

An *alternating Turing machine* on words over an alphabet Σ is a Turing machine $M = (Q, Q_\exists, Q_\forall, q_{in}, q_f, \delta)$, where $Q_\exists$ and $Q_\forall$ are disjoint sets of respectively existential and universal states that form a partition of Q, q_{in} is the initial state, q_f is the final state and $\delta : Q \times \Sigma \times \{D_1, D_2\} \longrightarrow Q \times \Sigma \times \{L, R\}$. For each pair $(q, \sigma) \in Q \times \Sigma$, there are exactly two transitions that we denote respectively as the D_1-transition and the D_2-transition. Suppose q is the current state and the tape head is reading the symbol σ on cell i, if the d-transition $\delta(q, \sigma, d) = (q', \sigma', L)$ is taken, M writes σ' on cell i, enters state q' and moves the read head to the left (L) to cell $(i-1)$. A configuration of M is a word $\sigma_1 \ldots \sigma_{i-1}(q, \sigma_i) \ldots \sigma_n$ where $\sigma_1 \ldots \sigma_n$ is the content of the tape and where M is at state q with the tape head at cell i. The *initial configuration* contains the word w and the initial state. An *outcome* of M is a sequence of configurations, starting from the initial configuration, constructed as a play in the game where the $\exists$-player picks the next transition (i.e. D_1 or D_2) when the play is in a state of $Q_\exists$, and the $\forall$-player picks the next transition when the play is in a state of $Q_\forall$. A *computation* of M is a strategy of the $\exists$-player, and an input word w is accepted iff there exists a computation such that all plays according to it reach a configuration with state q_f. We recall that an alternating Turing machine is $g(n)$ time-bounded if it halts on all input words of length n within $g(n)$ steps, and $g(n)$ space-bounded if it halts on all input words of length n using at most $g(n)$ tape cells (see [9]). We also recall that the acceptance problem is EXPSPACE-complete for exponentially time-bounded alternating Turing machines, and is 2EXPTIME-complete for exponentially space-bounded alternating Turing machines [5].

3.2 The matching trick

Fix n (which we assume is a power of 2) and let $m = \log_2 n$. Let us fix a set of propositions $\{p_1^\top, \ldots, p_m^\top, p_1^\perp, \ldots p_m^\perp\}$ and another set $\{q_1^\top, \ldots, q_m^\top, q_1^\perp, \ldots q_m^\perp\}$. Let us also fix a finite alphabet Σ disjoint from these. The gadget we describe allows us to access the i^{th} element of a sequence of n letters over Σ, using a formula which is polynomial in log n and which uses only the $\Diamond$ modality. Let $[i, j]$ denote the set of numbers from i to j, both inclusive. Let $[i]$ denote $[1, i]$.

Let $\boldsymbol{u} = u_m \ldots u_1$ denote a sequence of length m such that u_b is either $\{p_b^\top\}$ or $\{p_b^\perp\}$, for $b \in [m]$. Similarly, let $\boldsymbol{v} = v_1 \ldots v_m$ (note the reversal in indices) denote a sequence where each v_b is either $\{q_b^\top\}$ or $\{q_b^\perp\}$. We call these sequences u-addresses and v-addresses, respectively. A u-address $\boldsymbol{u} = u_m \ldots u_1$ is to be seen as the binary representation of a number from 0 to $n-1$: the proposition $p_b^\top$ belongs to u_b if the b^{th} bit of the number is 1, otherwise $p_b^\perp$ belongs to u_b (u_m encodes the most-significant bit). Similarly, a v-address $\boldsymbol{v}$ also represents a number, but note that the representation is reversed with the most-significant bit v_m at the end of the sequence. For $i \in \{0, \ldots, n-1\}$, let $u[i]$ and $v[i]$ denote the u-address and v-address representing the number i. For example, if $m = 4$, $u[5] = \{p_4^\perp\} \cdot \{p_3^\top\} \cdot \{p_2^\perp\} \cdot \{p_1^\top\}$ and $v[5] = \{q_1^\top\} \cdot \{q_2^\perp\} \cdot \{q_3^\top\} \cdot \{q_4^\perp\}$.

We encode a letter $a \in \Sigma$ at a position $i \in [0, n-1]$ as the string $\langle a \rangle_i = u[i] \cdot a \cdot v[n-1-i]$, i.e., $\langle a \rangle_i$ has 'a' sandwiched between a u-address representing i and a v-address representing $n-1-i$. Note that in $\langle a \rangle_i$, $v_b = \{q_b^\top\}$ iff $u_b = \{p_b^\perp\}$, for every $b \in [m]$. Now consider a sequence of such encodings in ascending order $\langle a_0 \rangle_0 \cdot \langle a_1 \rangle_1 \cdot \ldots \cdot \langle a_{n-1} \rangle_{n-1}$. We call such a sequence a *proper* sequence. Note that while the u-addresses run from 0 to $n-1$, the v-addresses run from $n-1$ to 0. Let w be a proper sequence, $a \in \Sigma$ and $i \in [0, n-1]$. We say that *a matches i in w* if $\langle a \rangle_i$ is a (not necessarily contiguous) subsequence of w. The main property of this encoding is that it allows us to check whether a letter a is encoded at a position i by simply checking whether $\langle a \rangle_i$ is a subsequence of the proper sequence w, i.e. by checking if a matches i in w. For example, when $m = 4$, consider $w = u[0]a_0v[15] \ldots u[15]a_{15}v[0]$. Let us consider for which letters $a \in \Sigma$, the string $u[5]av[10]$ is a subsequence of w. $\{p_3^\top\}$ is in $u[5]$ and the first place where it occurs in w is in the address $u[4]$. After this, the first place where $\{p_1^\top\}$ occurs in w is in $u[5]$. Hence the shortest prefix w' of w such that $u[5]$ is a subsequence of w is $u[0]a_0v[15]\ldots u[5]$. Similarly the shortest suffix w'' of w such that $v[10]$ is a subsequence of w'' is $v[10]\ldots u[15]a_{15}v[0]$. Hence if $u[5]av[10]$ is a subsequence of w, then $a = a_5$. The following lemma captures this:

Lemma 1. *Let $w = \langle a_0\rangle_0 \cdot \langle a_1\rangle_1 \cdot \ldots \cdot \langle a_{n-1}\rangle_{n-1}$ be a proper sequence, $a \in \Sigma$ and $i \in [0, n-1]$. Then, a matches i in w iff $a = a_i$.*

Finally, we show how to check whether a matches i in w using a formula in $\mathcal{B}(L_{\diamond,\wedge,\vee}(\Pi))$. If $\beta_1, \ldots, \beta_k$ are state predicates, let $\mathit{Seq}(\beta_1, \ldots, \beta_k)$ stand for the formula $\diamond(\beta_1 \wedge \diamond(\beta_2 \wedge \ldots \diamond(\beta_k)\ldots))$. Intuitively, this checks if there is a subsequence along which β_1 through β_k hold, in that order. Let $a \in \Sigma$ and $i \in [0, n-1]$. Let $x_m, \ldots, x_1$ be such that $x_b = \mathit{true}$ iff the b^{th} bit in the binary representation of i is 1. Let $\mathit{Same}(p_b, x_b)$ stand for the formula $(p_b^\top \wedge x_b) \vee (p_b^\perp \wedge \neg x_b)$, which asserts that the value of p_b is the same as that of x_b, where $b \in [m]$. Similarly, let $\mathit{Diff}(q_b, x_b)$ be the formula $(q_b^\top \wedge \neg x_b) \vee (q_b^\perp \wedge x_b)$, which asserts that the value of q_b is the negation of x_b. With $\mathit{Match}(a, i)$ we denote the formula $\mathit{Seq}(\mathit{Same}(p_m, x_m), \ldots, \mathit{Same}(p_1, x_1), a, \mathit{Diff}(q_1, x_1), \ldots, \mathit{Diff}(q_m, x_m))$. It is then easy to see that for any proper sequence w, w satisfies $\mathit{Match}(a, i)$ iff $\langle a\rangle_i$ is a subsequence of w, i.e iff a matches i in w.

3.3 Lower bound for $\mathcal{B}(L_{\diamond,\wedge,\vee}(\Pi))$

We show in this section that deciding $\mathcal{B}(L_{\diamond,\wedge,\vee}(\Pi))$ games is EXPSPACE-hard. The reduction is from the membership problem for alternating exponential-time Turing machines. We show that for such a Turing machine M, given an input word w, we can construct an instance of a game and a $\mathcal{B}(L_{\diamond,\wedge,\vee}(\Pi))$ specification in polynomial time such that M accepts w if and only if the protagonist has a winning strategy in the game.

Let us fix an alternating exponential-time Turing machine $M = (\Sigma, Q, Q_\exists, Q_\forall, q_{in}, q_f, \delta)$ and an input word $w \in \Sigma^*$. Let us assume that M takes at most m units of time on w (m is exponential in $|w|$). The game we construct will be such that the protagonist generates sequences of configurations beginning with the initial configuration (with w on its tape). During the game, after an existential configuration the protagonist gets to choose the transition (i.e. D_1 or D_2) while at a universal configuration the adversary gets to pick the transition. The specification will demand that successive configurations do indeed correspond to moves of M. Hence strategies of the protagonist correspond to runs on w.

Let π be any play according to a computation. Then π is a sequence of configurations of length m and each configuration can be represented using a sequence of at most m symbols (since M cannot use space more than m). We record π by encoding each cell of each configuration by explicitly encoding the number of the configuration and the number of the cell within the configuration. Configurations are represented as strings over the alphabet $\Sigma' = \Sigma \cup (Q \times \Sigma)$, namely from the set $\Sigma^* \cdot (Q \times \Sigma) \cdot \Sigma^*$.

In order to describe the configuration number and the cell number, each of which ranges from 0 to $m-1$, we need k bits where $m^2 = 2^k$. Let us fix a set of p-bits $\{p_l^z \mid l \in [1, k], z \in \{\top, \perp\}\}$ and a set of q-bits $\{q_l^z \mid l \in [1, k], z \in \{\top, \perp\}\}$.

We employ the matching trick using these p-bits, q-bits and Σ' (see Section 3.2 and recall the definitions).

For any $i \in [0, m^2-1]$, the $k/2$ less significant bits of i will represent the cell number and the $k/2$ more significant bits will represent the configuration number. Let $u'[\mathit{conf}, \mathit{cell}]$, where $\mathit{conf}, \mathit{cell} \in [0, m-1]$ denote the u-address $u[2^{k/2}.\mathit{conf} + \mathit{cell}]$. Hence $u'[\mathit{conf}, \mathit{cell}]$ encodes that the current configuration number is conf and the current cell number is cell. Similarly, define $v'[\mathit{conf}, \mathit{cell}]$ as the v-address $v[2^{k/2}.\mathit{conf} + \mathit{cell}]$. A proper sequence hence is of the form:

$$u'[0,0]a_{(0,0)}v'[m-1,m-1] \ldots u'[0,m-1]a_{(0,m-1)}v'[m-1,0]\ u'[1,0]a_{(1,0)}v'[m-2,m-1] \ldots\ldots\ldots\ldots u'[m-1,m-1]a_{(m-1,m-1)}v'[0,0].$$

The game graph we construct is composed of three parts: a main part and two sub-graphs G_1 and G_2. In the main part, the protagonists aim is to generate sequences of letters from Σ' sandwiched between a u-address and a v-address that form a proper sequence. The adversary's aim is to check that the current sequence is proper and conforms to the behavior of M. If the adversary claims that one of these is not true, the game moves to the subgraphs G_1 and G_2 where the adversary will have to provide witnesses to prove these claims.

The set of propositions we use includes the p-bits, q-bits and those in Σ', as well as a set of r-bits, s-bits, t-bits and e-bits $\{r_b^z, s_b^z, t_b^z, e_b^z \mid b \in [1,k], z \in \{\top, \bot\}\}$, and other new propositions $\{D_1, D_2, ok, obj_1, obj_2\}$. The game graph typically allows plays that look like:

$$u_0a_0v_0\ \overset{\nearrow\, obj_1}{ok} \ldots\ldots u_ya_yv_y\ \overset{\nearrow\, obj_1}{ok}\ d\ u'_0a'_0v'_0 \ldots\ldots\ldots u_fa_fv_f\ \underset{\searrow\, obj_2}{\overset{\nearrow\, obj_1}{ok}}\ ok\ ok \ldots$$

The central line above shows how a play normally proceeds. The protagonist generates sequences of triples consisting of a u-address, a letter in Σ' and a v-address. Note that each triple has $2k+1$ letters. After every such triple, the adversary gets a chance where it can continue the normal course by choosing ok or can generate an objection by choosing obj_1. Objection obj_1 leads the play to the subgraph G_1. Whenever the protagonist generates an address where the last $k/2$ bits of the u-address is 1 (denoted $u_ya_yv_y$ above), this denotes the end of a configuration, and the play reaches a protagonist state if the configuration generated was existential and an adversary state if it was universal. Accordingly, the protagonist or the adversary choose the next letter $d \in \{D_1, D_2\}$ which denotes the transition they wish to take from the current configuration.

At the end of the whole sequence, when a u-address with all its k bits set to 1 is generated (denoted $u_fa_fv_f$ above), the adversary, apart from being able to raise the first kind of objection, can also raise a second kind of objection by choosing the action obj_2 that leads the play to the sub-graph G_2. If the adversary instead chooses not to raise an objection, the game enters a state where the action ok occurs infinitely often and no other actions are permitted.

Note that the game graph does not ensure that the sequence generated is a proper sequence; in fact, it even allows triples of the form $u[i] \cdot a \cdot v[j]$ where $j \neq 2^k-1-i$. The objections obj_1 and obj_2 will take care of this and make sure that a proper sequence needs to be generated for the protagonist to win.

On the objection obj_1, the play moves to G_1 where the adversary claims that in the sequence generated thus far, there was a u-address for i which was not followed by a u-address for $i+1$, or there was a u-address for i which was not followed by a v-address for 2^k-1-i. The adversary chooses as a witness a sequence of k r-bits $x_k \ldots x_1$, where $x_b = \{r_b^\top\}$ or $x_b = \{r_b^\perp\}$. This denotes the binary representation of a number $\bar{r}$, with $x_b = \{r_b^\top\}$ iff the b^{th} bit in the binary representation of $\bar{r}$ is 1 (with x_k encoding the most significant bit). Next, the adversary chooses a sequence of k s-bits, encoding a number $\bar{s}$. The adversary should choose these sequences such that $\bar{s} = \bar{r}+1$.

The fact that $\bar{s} = \bar{r}+1$ can be checked using the formula $succ(\bar{r}, \bar{s})$:

$$\bigvee_{j=1}^{k} \left(\bigwedge_{h=1}^{j-1} (\Diamond r_h^\top \wedge \Diamond s_h^\perp) \wedge (\Diamond r_j^\perp \wedge \Diamond s_j^\top) \wedge \bigwedge_{h=j+1}^{k} ((\Diamond r_h^\top \wedge \Diamond s_h^\top) \vee (\Diamond r_h^\perp \wedge \Diamond s_h^\perp)) \right)$$

Let $same(p_i, r_i)$ stand for the formula $((p_i^\top \wedge \Diamond r_i^\top) \vee (p_i^\perp \wedge \Diamond r_i^\perp))$. Similarly define $same(p_i, s_i)$. Also, let $diff(q_i, r_i)$ stand for the formula $((q_i^\top \wedge \Diamond r_i^\perp) \vee (q_i^\perp \wedge \Diamond r_i^\top))$, which checks if the i^{th} q-bit is the complement of the i^{th} r-bit. The specification formula then has the following conjunct ψ_1:

$$\begin{aligned}
\psi_1 &= \Diamond obj_1 \rightarrow ([(succ(\bar{r}, \bar{s}) \wedge \varphi_1) \rightarrow \varphi_2] \wedge (\varphi_1' \rightarrow \varphi_2')) \;\; where \\
\varphi_1 &= Seq(same(p_k, r_k), \ldots, same(p_1, r_1), (p_1^\top \vee p_1^\perp)) \\
\varphi_2 &= Seq(same(p_k, r_k), \ldots, same(p_1, r_1), same(p_k, s_k), \ldots, same(p_1, s_1)) \\
\varphi_1' &= Seq(same(p_k, r_k), \ldots, same(p_1, r_1)) \\
\varphi_2' &= Seq(same(p_k, r_k), \ldots, same(p_1, r_1), diff(q_1, r_1), \ldots, diff(q_k, r_k))
\end{aligned}$$

φ_1 says that there is a subsequence of p-bits matching $\bar{r}$ and after this matching there is a future point where some p-bit (and hence a u-address) is defined, i.e. $u[\bar{r}].p_1^\top$ or $u[\bar{r}].p_1^\perp$ is a subsequence of the play. φ_2 demands that there is a subsequence of p-bits that match $\bar{r}$ followed by a sequence of p-bits matching $\bar{s}$, i.e. $u[\bar{r}] \cdot u[\bar{s}]$ is a subsequence.

The formula φ_1' checks whether there is a subsequence of p-bits matching $\bar{r}$ and φ_2' checks if there is a subsequence of p-bits matching r followed by a subsequence of q-bits matching $2^k-1-\bar{r}$, i.e. whether $u[\bar{r}] \cdot v[2^k-1-\bar{r}]$ is a subsequence of the play.

Consider a strategy for the protagonist such that all plays according to the strategy satisfy ψ_1. If the sequence $u[i_0]a_0v[j_0]$ ok $u[i_1]a_1v[j_1]$ ok $\ldots$ is a play according to this strategy, we can prove that this must be a proper sequence. One can also show that if the protagonist plays only proper sequences, then she cannot lose because of the conjunct ψ_1.

In summary, we have so far managed to construct a game graph that lets the protagonist generate cell contents at various addresses ranging from 0 to m^2-1. The specification ψ_1 forces the protagonist to generate this in increasing order. The game graph forces each contiguous block of declared cells to be a configuration and when a configuration is finished, the game graph ensures that the correct player gets to choose the direction of how the computation will proceed.

Let us now turn to the second objection obj_2 that ensures that the configuration sequences generated do respect the transitions of the Turing machine. After raising objection obj_2, the play reaches the subgraph G_2, where the adversary picks four numbers $\bar{r}$, $\bar{s}$, $\bar{t}$ and $\bar{e}$ (using the r-, s-, t- and e-bits). The adversary should generate these such that $\bar{s} = \bar{r} + 1$, $\bar{t} = \bar{s} + 1$ and $\bar{e} = \bar{s} + 2^{k/2}$. Also, $\bar{r}$, $\bar{s}$ and $\bar{t}$ will point to three consecutive cells of a particular configuration and hence $\bar{e}$ will point to a cell in the *successor configuration* where the cell number is the same as that pointed to by $\bar{s}$. The adversary claims that the cell content defined at $\bar{e}$ is not correct (note that the cell content at $\bar{e}$ solely depends on the cell contents at $\bar{r}$, $\bar{s}$ and $\bar{t}$).

First, the correctness of the values of $\bar{r}$, $\bar{s}$, $\bar{t}$ and $\bar{e}$ can be ensured using a formula φ_3, similar to the way we check whether a number is the successor of another. Also, let Δ_{D_1} be the set of all elements of the form $\langle a_1, a_2, a_3, a_2' \rangle$, where $a_1, a_2, a_3, a_2' \in \Sigma'$ and a_2' is the expected value of the cell number *cell*, if a_1, a_2 and a_3 are the cell contents of the cells $(\mathit{cell}-1)$, *cell* and $(\mathit{cell}+1)$ of the previous configuration and the machine took the D_1 transition. Similarly, define Δ_{D_2}. Let $\mathit{match}(a, r, \varphi)$, where φ is a temporal formula, denote the formula which checks whether a matches $\bar{r}$ in w (where $\bar{r}$ is the number encoded by the r-bits that occur somewhere in the future of where the string is matched) and after matching, the suffix of w from that point satisfies φ. More precisely, let $\mathit{same}(p_b, r_b) = ((p_b^\top \wedge \Diamond r_b^\top) \vee (p_b^\perp \wedge \Diamond r_b^\perp))$, as defined before, and let $\mathit{diff}(q_b, r_b) = ((q_b^\top \wedge \Diamond r_b^\perp) \vee (q_b^\perp \wedge \Diamond r_b^\top))$, for every $b \in [1, k]$. Then, $\mathit{match}(a, r, \varphi) = \mathit{Seq}(\mathit{same}(p_k, r_k), \ldots, \mathit{same}(p_1, r_1), a, \mathit{diff}(q_1, r_1), \ldots, \mathit{diff}(q_k, r_k), \varphi)$. Note that if φ is in $\mathcal{B}(L_{\Diamond,\wedge,\vee}(\Pi))$, then so is $\mathit{match}(a, r, \varphi)$. Define similarly formulas for s, t and e.

Now, we have in the specification the conjunct $\psi_2 = (\Diamond \mathit{obj}_2 \wedge \varphi_3) \to \varphi_4$, where φ_3 is as explained earlier and, denoting by $\xi(a_1, a_2, a_3, a_2', d)$ the formula $\mathit{match}(a_1, r, \mathit{match}(a_2, s, \mathit{match}(a_3, t, \Diamond(d \wedge \mathit{match}(a_2', e, \ \mathit{true})))))$,

$$\varphi_4 = \bigvee_{d \in \{D_1, D_2\}, \langle a_1, a_2, a_3, a_2' \rangle \in \Delta_d} \xi(a_1, a_2, a_3, a_2', d).$$

φ_4 checks whether there is tuple $\langle a_1, a_2, a_3, a_2' \rangle$ in Δ_{D_1} or Δ_{D_2} such that a_1 matches $\bar{r}$ followed by a_2 matching $\bar{s}$ followed by a_3 matching $\bar{t}$ followed by the corresponding direction D_1 or D_2 followed by a_2' matching $\bar{e}$. It is easy to see that a proper sequence that encodes a list of valid configurations interspersed with direction labels satisfies ψ_2 (for all possible values of $\bar{r}$, $\bar{s}$, $\bar{t}$ and $\bar{e}$) iff it corresponds to a correct evolution of M according to the direction labels.

The complete specification is then $\psi_1 \wedge \psi_2 \wedge \psi_3$, where $\psi_3 = \Diamond \mathit{obj}_1 \vee \Diamond \mathit{obj}_2 \vee \bigvee_{a \in \Sigma} \Diamond(q_f, a)$ which demands that if no objection is raised, then the play must meet the final state.

We can show thus that there is a winning strategy for the protagonist iff M accepts w. Since the main part of G is $O(|w| + k + |M|)$, size of both G_1 and G_2 is $O(k)$, and size of $\psi_1 \wedge \psi_2 \wedge \psi_3$ is $O(k\,(k + |M|))$, we have:

Theorem 1. *Deciding $\mathcal{B}(L_{\Diamond,\wedge,\vee}(\Pi))$ games is* EXPSPACE*-hard.*

3.4 Lower bound for $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$

In this section, we show that deciding games for specifications given by formulas in $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ is 2EXPTIME-hard. The reduction is from the membership problem for alternating exponential-space Turing machines. We show that for such a Turing machine M, given an input word w, we can construct in polynomial time a game graph G' and an $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ formula φ such that M accepts w if and only if the protagonist has a winning strategy in the game.

Let G be the game graph constructed in Section 3.3. We give a reduction based on the construction of a game graph G' which is slightly different from G. First, the configuration numbers are not encoded explicitly (since they can be doubly exponential) but the cell numbers are encoded. However, for a configuration sequence $c_0 c_1 \ldots$, we want to count the configurations using a counter modulo 3. For this, we introduce new propositions $\mathcal{P}' = \{0, 1, 2\}$, and in the entire sequence encoding c_i, the proposition $i \ mod \ 3$ is true. This counter's behaviour is ensured by the design of the game graph.

The role of obj_1 is similar as in G; using this the adversary ensures that the sequence generated is proper and hence that the system does generate a sequence of configurations with proper cell numbers. However, if it wants to claim that the sequence of configurations is not according to the Turing machine, note that it cannot provide an exact witness as configurations are not numbered. We hence allow the adversary to raise the objection obj_2 after the end of *every* configuration. When the adversary chooses obj_2 it gives a cell number k and claims that the contents of cell k in the *last* configuration thus far is incorrect with respect to the corresponding cells in the previous configuration.

The crucial point is that one can check whether the cell in the last configuration is correct with respect to the penultimate configuration by using an $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ formula that uses the modulo-3 counter. Intuitively, if we want to check a formula φ_1 on the suffix starting from the penultimate configuration, we can do so by the formula

$$\bigvee_{j \in \{0,1,2\}} (\Diamond (j \wedge \varphi_1 \wedge \Diamond(j+1) \wedge \neg\Diamond(j+2)))$$

Note that the formula is in $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$, but not in $\mathcal{B}(L_{\Diamond,\wedge,\vee}(\Pi))$, because of the subformula $\neg\Diamond(j+2)$ (which plays a vital role). Using such a formula, we ensure that the protagonist must generate correct configuration sequences to win the game. The rest of the proof is in details and we omit them; we then have:

Theorem 2. *Deciding $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ games is 2*EXPTIME*-hard.*

4 Fairness games

In modeling reactive systems, fairness assumptions are added to rule out infinite computations in which some choice is repeatedly ignored [11]. A typical fairness constraint is of the form "if an action is enabled infinitely often, then it is taken infinitely often," and is captured by a formula of the form $\Box\Diamond p \to \Box\Diamond p'$. In the game setting, fairness constraints can refer to both the players. Let ψ_0 be the formula expressing the fairness constraint for the protagonist and ψ_1 be the formula for the fairness constraints of the adversary. Then, the winning condition φ of a game is changed either to $\psi_1 \to (\psi_0 \wedge \varphi)$ ("if the adversary is fair then the protagonist plays fair and satisfies the specification") or $\psi_0 \wedge (\psi_1 \to \varphi)$ ("the protagonist plays fair and if the adversary plays fair then the specification is satisfied") [3]. Thus, adding fairness changes the winning conditions for specifications from a logic $\mathcal{L}$ to Boolean combinations of $L_{\Box\Diamond}(\Pi)$ and $\mathcal{L}$ formulas.

We consider adding fairness to $\mathcal{B}(L_{\Diamond,\wedge}(\Pi))$ games. The fragment $\mathcal{B}(L_{\Diamond,\wedge}(\Pi))$ contains Boolean combinations of formulas built from state predicates using eventualities and conjunctions, and includes combinations of typical invariants and termination properties. A sample formula of this fragment is $\Box p \vee \Diamond(q \wedge \Diamond r)$.

In this section, we prove that games for this logic augmented with fairness constraints are still decidable in polynomial space. More precisely, we prove that deciding $\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond,\wedge}(\Pi))$ games is PSPACE-complete. We begin by considering $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ games.

4.1 Boolean combinations of Büchi conditions

In this section we give a polynomial space algorithm to solve $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ games. We adapt the technique proposed by Zielonka [17] for Muller games. Muller games are game graphs with winning conditions given as a collection of sets of vertices $\mathcal{F}$ with the meaning that a play π is winning if the set of the infinitely repeating vertices in π belongs to $\mathcal{F}$.

Let V be a finite set, $\mathcal{F}$ be a subset of 2^V, and $\bar{\mathcal{F}}$ be $2^V \setminus \mathcal{F}$. A set $U \in \mathcal{F}$ is maximal for $\mathcal{F}$ if for all $U' \in \mathcal{F}$, $U \not\subset U'$. A *Zielonka tree* for the pair $(\mathcal{F}, \bar{\mathcal{F}})$ is a finite tree T with vertices labeled by pairs of the form $(0, U)$ with $U \in \mathcal{F}$ or $(1, U)$ with $U \in \bar{\mathcal{F}}$. It is inductively defined as follows. The root of T is labeled with $(0, V)$, if $V \in \mathcal{F}$, and by $(1, V)$ otherwise. Suppose x is a node labeled with $(0, U)$. If $U_1, \ldots, U_m$, $m > 0$, are the maximal subsets of U belonging to $\bar{\mathcal{F}}$, then x has m children respectively labeled with $(1, U_1), \ldots, (1, U_m)$. If all subsets of U belong to $\mathcal{F}$, then x is a leaf. The case $(1, U)$ is analogous. Notice that while the number of children can be exponential in $|V|$, the depth of the tree is linear in $|V|$.

Z-solve(G_x, x)
  Let V_x be the set of G_x vertices and let (i, U_x) be the label of x
  $W \leftarrow W' \leftarrow \emptyset$
  if x is not a leaf **then**
    repeat
      $W \leftarrow W \cup W'$; $W' \leftarrow \emptyset$
      $W \leftarrow$ *Attractor-set*$(W, 1-i)$
      for each child y of x **do**
        Let $(1-i, U_y)$ be the label of y
        $G_y \leftarrow$ *Sub-game*(G_x, W, U_y)
        $W' \leftarrow W' \cup$ *Z-solve*(G_y, y)
    until $(W = W \cup W')$
  return $(V_x \setminus W)$

Fig. 1. Algorithm for $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ games.

Let V be the set of vertices of a game graph G and $\mathcal{F}$ denote a Muller winning condition. The algorithm in Figure 1 implements the solution given by Zielonka [17]. Let G_x be a sub-game of G (i.e. G_x is a game graph which is a subgraph of G) and let x be a node of the Zielonka tree for $(\mathcal{F}, \bar{\mathcal{F}})$, with x labeled with (i, U_x). On the call *Z-solve*(G_x, x), the procedure computes the set of positions in G_x from which player i has a winning strategy with respect to the Muller condition restricted to G_x. The procedure is hence initially invoked with $(G, \hat{x})$ where $\hat{x}$ is the root of the Zielonka tree.

The procedure works by growing the set W of the vertices from which player $1-i$ has a winning strategy on G_x. The main parts of *Z-solve*(G_x, x) are the enumeration of the children of x, and the calls to procedures *Attractor-set* and *Sub-game*. A call *Attractor-set*$(G_x, W, 1-i)$ constructs the largest set of vertices in G_x from which player $1-i$ has a strategy to reach W. For a set $U \subseteq V_x$, let Z be the set of vertices constructed in the call *Attractor-set*$(V \setminus U, i)$. Then, *Sub-game*(G_x, W, U) constructs a game graph contained in G_x, that is induced by the vertices $V_x \setminus (W \cup Z)$. Each call to either *Attractor-set* or *Sub-game* takes at most polynomial time. Note that the recursive depth of calls is bounded by the depth of the tree.

It is worth noting that for an implicitly defined Muller condition, such as the one defined by a formula in $\mathcal{B}(L_{\Box\Diamond}(\Pi))$, one can use the same procedure above but without explicitly constructing the Zielonka tree. The algorithm just needs to compute the children of a node of the tree, and, as we show below, this can be done in polynomial space as long as the membership test $U \in \mathcal{F}$ can be implemented in polynomial space. Note that the recursive call depth is bounded by V and hence the algorithm will run in polynomial space.

To explain the computation of children of a node of the Zielonka tree more formally, consider a $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ formula φ. For a set $U \subseteq V$, let ν_U be the mapping that assigns a sub-formula $\Box\Diamond p$ of φ to true if p holds in some vertex in U, and assigns it to false otherwise. We say that U *meets* φ if under the assignment ν_U, φ evaluates to true. Intuitively, if U is the set of the vertices that repeat infinitely often on a play π of G, then π satisfies φ if and only if U meets φ. Let $\mathcal{F}_\varphi$ be the set of $U \subseteq V$ such that U meets φ, and $\bar{\mathcal{F}}_\varphi$ be its complement with respect to 2^V. The Zielonka tree for φ is the Zielonka tree T for $(\mathcal{F}_\varphi, \bar{\mathcal{F}}_\varphi)$. We observe that each child of a node x of T can be generated in polynomial space simply from φ and the label (i, U) of x. For example, if $i = 0$, then for each $U' \subseteq U$ we can check if $(1, U')$ is a child of x by checking whether it falsifies φ and is maximal within the subsets of U that do not meet φ (which can be done in polynomial space). Thus we can enumerate the children of a node in the Zielonka tree using only polynomial space and we have the following:

Theorem 3. *Deciding $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ games is in* PSPACE.

4.2 Solving fairness games

The result from Section 4.1 can be extended to prove that when the winning condition is a formula from $\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond,\wedge}(\Pi))$, that is a Boolean combination of formulas from $L_{\Box\Diamond}(\Pi)$ and $L_{\Diamond,\wedge}(\Pi)$, then games can still be decided in polynomial space.

We first describe a polynomial space algorithm to solve games for the simpler fragment of the Boolean combinations of $L_{\Box\Diamond}(\Pi)$ formulas and formulas of the form $\Diamond p$ using the polynomial-space algorithm for $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ above.

Let us explain the intuition behind the solution with an example. Consider the formula $\varphi = \Box\Diamond p_1 \vee (\Box\Diamond p_2 \wedge \Diamond p_3)$. The protagonist can win a play by visiting a state satisfying p_1 infinitely. However, if it meets a state satisfying p_3 then it wins using the formula $\varphi' = \Box\Diamond p_1 \vee \Box\Diamond p_2$. Now, assume that we know the exact set Z of positions from which the protagonist can win the game G with

φ' as the winning condition. We construct a game graph G' from G by adding two vertices *win* and *lose*, that have self loops and let a new proposition p^{win} be true only at *win*. Now, for a vertex u in G where p_3 holds, we remove all the edges from u and instead add an edge to either *win* or *lose* — we add an edge to *win* if u is in Z, and an edge to *lose* otherwise. Then clearly the protagonist wins the overall game if and only if it wins the game $(G', \Box\Diamond p_1 \vee \Box\Diamond p^{win})$. In general, we need to define and solve many such games. Each such game corresponds to some subset X of the subformulas of the kind $\Diamond p_i$ mentioned in φ. In such a game, when we meet a state that meets a new predicate p, where $\Diamond p$ is a subformula of φ but is not in X, we jump to *win* or *lose* depending on whether the game corresponding to $X \cup \{\Diamond p\}$ is winning or losing.

Consider a $\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond}(\Pi))$ game (G, φ) and let $\Diamond p_1, \ldots, \Diamond p_k$ be all the sub-formulas of φ of the form $\Diamond p$ that are not in the scope of the $\Box$ operator. For each assignment ν of truth values to $\Diamond p_1, \ldots, \Diamond p_k$, define φ_ν as the formula obtained from φ by assigning $\Diamond p_1, \ldots, \Diamond p_k$ according to ν. Clearly, φ_ν is a $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ formula. For an assignment ν and a vertex u, we denote by $\nu + u$ the assignment that maps $\Diamond p_i$ to true iff either ν assigns $\Diamond p_i$ to true or p_i holds true at u. We say that a vertex u *meets* an assignment ν if whenever p_i holds true at u, then ν also assigns $\Diamond p_i$ to true.

We denote by G_ν the game graph obtained from G removing all the edges (u, v) such that u does not meet ν, and adding two new vertices *win* and *lose* along with new edges as follows. We use a new atomic proposition p^{win} that is true only at *win*. We add a self loop on both *win* and *lose*, and there are no other edges leaving from these vertices (i.e., they are both sinks). Denoting by X_ν the set of vertices that meet ν, we add an edge from each $u \notin X_\nu$ to *win* if there is a winning strategy of the protagonist in $(G_{\nu+u}, \varphi_{\nu+u} \vee \Box\Diamond p^{win}, u)$, and to *lose* otherwise.

In order to construct G_ν, we may need to make recursive calls to construct and solve $(G_{\nu+u}, \varphi_{\nu+u} \vee \Box\Diamond p^{win}, u)$, for some u. However, note that the number of elements set to true in $\nu + u$ is more than those set in ν to be true. Also, if ν assigns all elements to true, there will be no recursive call to construct games. Hence the depth of recursion is bounded by the number of subformulas of the kind $\Diamond p$ in φ. Note however that there can be an exponential number of calls required when constructing G_ν. Since any of the games (G_ν, φ'), once constructed, can be solved in polynomial space, it follows that $G_\perp$ can be constructed in polynomial space, where $\perp$ is the empty assignment where every element is set to false. Therefore, we have the following lemma.

Lemma 2. *Given a $\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond}(\Pi))$ game (G, φ, u), there exists a winning strategy of the protagonist in (G, φ, u) if and only if there exists a winning strategy of the protagonist in $(G_\perp, \varphi_\perp \vee \Box\Diamond p^{win}, u)$.*

To decide $\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond,\wedge}(\Pi))$ games we need to modify the above procedure. We know from [4] that for each formula ψ in $\mathcal{B}(L_{\Diamond,\wedge}(\Pi))$ there exists a deterministic Büchi automaton A accepting all the models of ψ such that: (1) size of A is exponential in $|\psi|$, (2) the automaton can be constructed "on-the-fly" — for any state of the automaton, the transitions from it can be found in polynomial time, (3) the length of simple paths in A is linear in $|\psi|$, and (4) the only cycles in the transition graph of A are the self-loops. For a given

$\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond,\wedge}(\Pi))$ formula φ, let $\psi_1, \ldots, \psi_k$ be all the sub-formulas of φ from $\mathcal{B}(L_{\Diamond,\wedge}(\Pi))$ that are not in the scope of the $\Box$ operator. Let A_i be a deterministic automaton accepting models of ψ_i and satisfying the above properties. Let Q be the product of the sets of states of $A_1, \ldots, A_k$. For a tuple $\bar{q} = \langle q_1, \ldots, q_k \rangle \in Q$, we associate a truth assignment $\nu[\bar{q}]$ such that $\nu[\bar{q}](\psi_i)$ is true if and only if q_i is an accepting state. Moreover, for $i = 1, \ldots, k$, let q'_i be the state entered by A_i starting from q_i reading the label of a vertex u of G. We denote the tuple $\langle q'_1, \ldots, q'_k \rangle$ by $\bar{q}(u)$. For a tuple of states $\bar{q} = \langle q_1, \ldots, q_k \rangle \in Q$, the graph $G_{\bar{q}}$ is constructed similar to G_ν. The main differences are that we solve games of the form $(G_{\bar{q}(v)}, \varphi_{\nu[\bar{q}(v)]} \vee \Box \Diamond p^{win}, v)$, we use the set $X_{\bar{q}} = \{v \mid \bar{q} \neq \bar{q}(v)\}$ instead of X_ν and the recursion depth is bounded by $O(k \cdot |\varphi|)$. Clearly, each such graph can be constructed in polynomial space and thus we have the following result.

Theorem 4. *Deciding $\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond,\wedge}(\Pi))$ games is in* PSPACE.

We can also show that deciding $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ games is PSPACE-hard and hence, from the results above, deciding $\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond,\wedge}(\Pi))$ games and $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ games is PSPACE-complete. We in fact show a stronger result that a fragment of $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ is already PSPACE-hard.

Let $\mathcal{L}_R$ denote the set of formulas of the form $\bigvee_{i=1}^{k} \varphi_i$, where each φ_i is of the form $(\Box \Diamond p_i \wedge \Diamond \Box p'_i)$ (where each p_i and p'_i are state predicates). $\mathcal{L}_R$ is then a fragment of $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ and represents Rabin conditions. Let $\mathcal{L}_S$ denote the set of formulas of the form $\bigwedge_{i=1}^{k} \varphi_i$, where each φ_i is of the form $(\Box \Diamond p_i \rightarrow \Box \Diamond p'_i)$. $\mathcal{L}_S$ is also a fragment of $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ and represents Streett conditions, which are the dual of Rabin conditions. Let $\mathcal{L}_{RS}$ represent a disjunction of a Rabin and a Streett condition, i.e. $\mathcal{L}_{RS}$ contains formulas of the kind $\varphi_R \vee \varphi_S$ where $\varphi_R \in \mathcal{L}_R$ and $\varphi_S \in \mathcal{L}_S$. We can then show the that deciding games for formulas in $\mathcal{L}_{RS}$ is PSPACE-hard. Note that it is known that deciding $\mathcal{L}_R$ games is NP-complete and, hence, deciding $\mathcal{L}_S$ games is CO-NP-complete.

Theorem 5. *Deciding $\mathcal{L}_{RS}$ games is* PSPACE*-hard.*

5 Conclusions

We have shown that games for the fragment $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ are 2EXPTIME-hard, games for the fragment $\mathcal{B}(L_{\Diamond,\wedge,\vee}(\Pi))$ are EXPSPACE-hard, games for $\mathcal{B}(L_{\Box\Diamond}(\Pi))$ are PSPACE-hard, and games for $\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond,\wedge}(\Pi))$ are in PSPACE. Our lower bound proofs introduce new techniques for counting and encoding using game graphs and LTL formulas without using next or until operators. Our upper bound techniques are useful for combinations of safety/reachability games on game graphs with strong fairness requirements on the choices of the two players. The results in this paper complete the picture for the complexity bounds for various fragments of LTL and is summarized in Figure 2. Recall that model-checking of $L_{\Box,\Diamond,\wedge,\vee}(\Pi)$ formulas is NP-complete, and becomes PSPACE-complete (as for the full LTL) by allowing the until and/or the next operators [15]. As shown in Figure 2, this does not hold for games. Note that allowing nested always and eventually operators the complexity of games increases to the complexity of the whole LTL (i.e., 2EXPTIME-complete), while the use of the next and eventually operators (with the negation only at the top level) that makes model checking PSPACE-hard, increases the complexity of games only to EXPSPACE.

	Lower bound	Upper bound
$\mathcal{B}(L_{\Diamond,\wedge}(\Pi))$	PSPACE-complete [4]	
$\mathcal{B}(L_{\Box\Diamond}(\Pi) \cup L_{\Diamond,\wedge}(\Pi))$	**Pspace-complete**	
$L_{\Diamond,\wedge,\vee}(\Pi)$	PSPACE-hard [4]	PSPACE [12]
$\mathcal{B}(L_{\Diamond,\bigcirc,\wedge}(\Pi))$	EXPTIME-complete [4]	
$\mathcal{B}(L_{\Diamond,\wedge,\vee}(\Pi))$	**Expspace-hard**	EXPSPACE [4]
$\mathcal{B}(L_{\Diamond,\bigcirc,\wedge,\vee}(\Pi))$	EXPSPACE-hard [12]	EXPSPACE [4]
$L_{\Box,\Diamond,\wedge,\vee}(\Pi)$	**2Exptime-hard**	2EXPTIME [14]
LTL	2EXPTIME-complete [14]	

Fig. 2. Complexity of LTL games.

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. of ICALP'89*, LNCS 372, pages 1–17, 1989.
2. R. Alur, L. de Alfaro, T. Henzinger, and F. Mang. Automating modular verification. In *Proc. of CONCUR'99*, LNCS 1664, pages 82–97, 1999.
3. R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):1–42, 2002.
4. R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *Proc. of LICS'01*, pages 291–302, 2001.
5. A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114 – 133, 1981.
6. D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits*. ACM Distinguished Dissertation Series. MIT Press, 1989.
7. E. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. of FOCS'88*, pages 328 – 337, 1988.
8. G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279 – 295, 1997.
9. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
10. O. Kupferman and M. Vardi. Module checking. In *Proc. of CAV'96*, LNCS 1102, pages 75 – 86. Springer, 1996.
11. Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer, 1991.
12. J. Marcinkowski and T. Truderung. Optimal complexity bounds for positive ltl games. In *Proc. of CSL 2002*, LNCS 2471, pages 262–275. Springer, 2002.
13. A. Pnueli. The temporal logic of programs. In *Proc. of FOCS'77*, pages 46 – 77, 1977.
14. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of POPL'89*, pages 179 – 190, 1989.
15. A. Sistla and E. Clarke. The complexity of propositional linear temporal logics. *The Journal of the ACM*, 32:733 – 749, 1985.
16. M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc. 17th Symp. on Theory of Computing*, pages 240 – 251, 1985.
17. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135 – 183, 1998.