Finite-state Strategies in Delay Games ☆

Sarah Winter[a], Martin Zimmermann[b]

[a] *Lehrstuhl für Informatik 7, RWTH Aachen University, 52056 Aachen, Germany*
[b] *University of Liverpool, Liverpool L69 3BX, United Kingdom*

Abstract

What is a finite-state strategy in a delay game? We answer this surprisingly non-trivial question by presenting a very general framework that allows to remove delay: finite-state strategies exist for all winning conditions where the resulting delay-free game admits a finite-state strategy. The framework is applicable to games whose winning condition is recognized by an automaton with an acceptance condition that satisfies a certain aggregation property.

Our framework also yields upper bounds on the complexity of determining the winner of such delay games and upper bounds on the necessary lookahead to win the game. In particular, we cover all previous results of that kind as special cases of our uniform approach.

Keywords: Delay Games, Finite-state Strategies, Transducers, Tradeoffs

1. Introduction

What is a finite-state strategy in a delay game? The answer to this question is surprisingly non-trivial due to the nature of delay games in which one player is granted a lookahead on her opponent's moves. This puts her into an advantage when it comes to winning games, i.e., there are games that can only be won with lookahead, but not without. A simple example is a game where one has to predict the third move of the opponent with one's first move. This is impossible when moving in alternation, but possible if

☆ Supported by the projects "TriCS" (ZI 1516/1-1) and (LO 1174/3-1) of the German Research Foundation (DFG). The work presented here was carried out while the second author was a member of the Reactive Systems Group at Saarland University.

Email addresses: winter@automata.rwth-aachen.de (Sarah Winter), martin.zimmermann@liverpool.ac.uk (Martin Zimmermann)

one has access to the opponent's first three moves before making the first move. More intriguingly, lookahead also allows to improve the quality of winning strategies in games with quantitative winning conditions, i.e., there is a tradeoff between quality and amount of lookahead [1]. More practically, when modeling reactive synthesis as a two-player game, the addition of delay allows us to model delay inherent to sensing and actuating in the physical world as well as the delay caused by the transmission of data [2]. Thus, delay games capture aspects of real-life synthesis problems that cannot easily be expressed in the classical, i.e., delay-free, framework.

However, managing (and, if necessary, storing) the additional information gained by the lookahead can be a burden. Consider another game where one just has to copy the opponent's moves. This is obviously possible with or without lookahead (assuming the opponent moves first). In particular, without lookahead one just has to remember the last move of the opponent and copy it. However, when granted lookahead, one has to store the last moves of the opponent in a queue to implement the copying properly. This example shows that lookahead is not necessarily advantageous when it comes to minimizing the memory requirements of a strategy.

In this work, we are concerned with Gale-Stewart games [3], abstract games without an underlying arena.[1] In such a game, both players produce an infinite sequence of letters and the winner is determined by the combination of these sequences. If it is in the winning condition, a set of such combinations, then the second player wins, otherwise the first one wins. In a classical Gale-Stewart game, both players move in alternation while in a delay game, the second player skips moves to obtain a lookahead on the opponent's moves. Which moves are skipped is part of the rules of the game and known to both players.

Delay games have recently received a considerable amount of attention after being introduced by Hosch and Landweber [4] only three years after the seminal Büchi-Landweber theorem [5]. Büchi and Landweber had shown how to solve infinite two-player games with ω-regular winning conditions. Forty years later, delay games were revisited by Holtmann, Kaiser, and Thomas [6] and the first comprehensive study was initiated, which settled many basic

[1]The models of Gale-Stewart games and arena-based games are interreducible, but delay games are naturally presented as a generalization of Gale-Stewart games. This is the reason we prefer this model here.

problems like the exact complexity of solving ω-regular delay games and the amount of lookahead necessary to win such games [7]. Furthermore, Martin's seminal Borel determinacy theorem [8] for Gale-Stewart games has been lifted to delay games [9] and winning conditions beyond the ω-regular ones have been investigated [10, 11, 12, 1].

Finally, the decision version of the uniformization problem is to decide whether a given relation has a uniformization, that is, whether there exists a function with a prescribed property that is contained in the relation and has the same domain. This problem for relations over infinite words and continuous functions boils down to solving delay games: a relation $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ is uniformized by a continuous function (in the Cantor topology) if, and only if, the delaying player wins the delay game with winning condition L. We refer to [6] for details.

What makes finite-state strategies in infinite games particularly useful and desirable is that a general strategy is an infinite object, as it maps finite play prefixes to next moves. On the other hand, a finite-state strategy is implemented by a transducer, an automaton with output, and therefore finitely represented: the automaton reads a play prefix and outputs the next move to be taken. Thus, the transducer computes a finite abstraction of the play's history using its state space as memory and determines the next move based on the current memory state.

In Gale-Stewart games, finite-state strategies suffice for all ω-regular games [5] and even for deterministic ω-contextfree games, if one allows pushdown transducers [13]. For Gale-Stewart games (and arena-based games), the notion is well-established and one of the most basic questions about a class of winning conditions is that about the existence and size of winning strategies for such games.

While foundational questions for delay games have been answered and many results have been lifted from Gale-Stewart games to those with delay, the issue of computing tractable and implementable strategies has not been addressed before. However, this problem is of great importance, as the existence and computability of finite-state strategies is a major reason for the successful application of infinite games to diverse problems like reactive synthesis, model-checking of fixed-point logics, and automata theory.

In previous work, restricted classes of strategies for delay games have been considered [9]. However, those restrictions are concerned with the amount of information about the lookahead's evolution a strategy has access to, and do not restrict the size of the strategies: In general, they are still infinite

objects. On the other hand, it is known that bounded lookahead suffices for many winning conditions of importance, e.g., the ω-regular ones [7], those recognized by parity and Streett automata with costs [1], and those definable in (parameterized) linear temporal logics [11]. Furthermore, for all those winning conditions, the winner of a delay game can be determined effectively. In fact, all these proofs rely on the same basic construction that was already present in the work of Holtmann, Kaiser, and Thomas, i.e., a reduction to a Gale-Stewart game using equivalence relations that capture behavior of the automaton recognizing the winning condition. These reductions and the fact that finite-state strategies suffice for the games obtained in the reductions imply that (some kind of) finite-state strategies exist for such games.

Indeed, in his master's thesis, Salzmann recently introduced the first notion of finite-state strategies in delay games and, using these reductions, presented an algorithm computing them for several types of acceptance conditions, e.g., parity conditions and related ω-regular ones [14]. However, the exact nature of finite-state strategies in delay games is not as canonical as for Gale-Stewart games. We discuss this issue in-depth in Sections 3 and 5 by proposing two notions of finite-state strategies, a delay-oblivious one which yields large strategies in the size of the lookahead, and a delay-aware one that follows naturally from the reductions to Gale-Stewart games mentioned earlier. In particular, the number of states of the delay-aware strategies is independent of the size of the lookahead, but often larger in the size of the automaton recognizing the winning condition. However, this is offset by the fact that strategies of the second type are simpler to compute than the delay-oblivious ones and have overall fewer states, if the lookahead is large. In comparison to Salzmann's notion, where strategies syntactically depend on a given automaton representing the winning condition, our strategies are independent of the representation of the winning condition and therefore more general. Also, our framework is more abstract and therefore applicable to a wider range of acceptance conditions (e.g., qualitative ones) and yields in general smaller strategies, but there are of course some similarities, which we discuss in detail.

To present these notions, we first introduce some definitions in Section 2, e.g., delay games and finite-state strategies for Gale-Stewart games. After introducing the two notions of finite-state strategies for delay games in Section 3, we show how to compute such strategies in Section 4. To this end, we present a generic account of the reduction from delay games to Gale-Stewart games which subsumes, to the best of our knowledge, all decidability results

presented in the literature. Furthermore, we show how to obtain the desired strategies from our construction. Then, in Section 5, we compare the different definitions of finite-state strategies for delay games proposed here and discuss their advantages and disadvantages. Also, we compare our approach to that of Salzmann. In Section 6, discuss how to implement finite-state strategies in delay games even more succinctly. We conclude by mentioning some directions for further research in Section 7.

This work is an extended and revised version of a paper presented at GandALF 2017 [15].

Related Work. As mentioned earlier, the existence of finite-state strategies is the technical core of many applications of infinite games, e.g., in reactive synthesis one synthesizes a correct-by-construction system from a given specification by casting the problem as an infinite game between a player representing the system and one representing the antagonistic environment. It is a winning strategy for the system player that yields the desired implementation, which is finite if the winning strategy is finite-state. Similarly, Gurevich and Harrington's game-based proof of Rabin's decidability theorem for monadic second-order logic over infinite binary trees [16] relies on the existence of finite-state strategies.[2]

These facts explain the need for studying the existence and properties of finite-state strategies in infinite games [17, 18, 19, 20]. In particular, the seminal work by Dziembowski, Jurdziński, and Walukiewicz [21] addressed the problem of determining upper and lower bounds on the size of finite-state winning strategies in games with Muller winning conditions. Nowadays, one of the most basic questions about a given winning condition is that about such upper and lower bounds. For most conditions in the literature, tight bounds are known, see, e.g., [22, 23, 24]. But there are also surprising exceptions to that rule, e.g., generalized reachability games [25]. More recently, Colcombet, Fijalkow, and Horn presented a very general technique that yields tight upper and lower bounds on memory requirements in safety games, which even hold for games in infinite arenas, provided their degree is finite [26].

[2]The proof is actually based on positional strategies, a further restriction of finite-state strategies for arena-based games, because they are simpler to handle. Nevertheless, the same proof also works for finite-state strategies.

2. Preliminaries

We denote the non-negative integers by $\mathbb{N}$. An alphabet Σ is a non-empty finite set. The set of finite words over Σ is denoted by Σ^* and the set of infinite words by Σ^ω. Given a finite or infinite word α, we denote by $\alpha(i)$ the ith letter of α, starting with 0, i.e., $\alpha = \alpha(0)\alpha(1)\alpha(2)\cdots$. Given two ω-words $\alpha \in (\Sigma_0)^\omega$ and $\beta \in (\Sigma_1)^\omega$, we define $\binom{\alpha}{\beta} = \binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)}\cdots \in (\Sigma_0 \times \Sigma_1)^\omega$. Similarly, we define $\binom{x}{y}$ for finite words x and y with $|x| = |y|$.

2.1. ω-automata

A (deterministic and complete) ω-automaton is a tuple $\mathfrak{A} = (Q, \Sigma, q_I, \delta, \mathrm{Acc})$ where Q is a finite set of states, Σ is an alphabet, $q_I \in Q$ is the initial state, $\delta\colon Q \times \Sigma \to Q$ is the transition function, and $\mathrm{Acc} \subseteq \delta^\omega$ is the set of accepting runs (here, and whenever convenient, we treat δ as a relation $\delta \subseteq Q \times \Sigma \times Q$). A finite run π of $\mathfrak{A}$ is a sequence

$$\pi = (q_0, a_0, q_1)(q_1, a_1, q_2)\cdots(q_{i-1}, a_{i-1}, q_i) \in \delta^+.$$

As usual, we say that π starts in q_0, ends in q_i, and processes $a_0 \cdots a_{i-1} \in \Sigma^+$. Infinite runs on infinite words are defined analogously. If we speak of *the* run of $\mathfrak{A}$ on $\alpha \in \Sigma^\omega$, then we mean the unique run of $\mathfrak{A}$ starting in q_I processing α. The language $L(\mathfrak{A}) \subseteq \Sigma^\omega$ of $\mathfrak{A}$ contains all those ω-words whose run of $\mathfrak{A}$ is accepting. The size of $\mathfrak{A}$ is defined as $|\mathfrak{A}| = |Q|$.

This definition is very broad, which allows us to formulate our theorems as general as possible. In examples, we consider safety, reachability, parity, and Muller automata whose sets of accepting runs are finitely represented: An ω-automaton $\mathfrak{A} = (Q, \Sigma, q_I, \delta, \mathrm{Acc})$ is a safety automaton, if there is a set $F \subseteq Q$ of accepting states such that

$$\mathrm{Acc} = \{(q_0, a_0, q_1)(q_1, a_1, q_2)(q_2, a_2, q_3)\cdots \in \delta^\omega \mid q_i \in F \text{ for every } i\}.$$

Moreover, an ω-automaton $\mathfrak{A} = (Q, \Sigma, q_I, \delta, \mathrm{Acc})$ is a reachability automaton, if there is a set $F \subseteq Q$ of accepting states such that

$$\mathrm{Acc} = \{(q_0, a_0, q_1)(q_1, a_1, q_2)(q_2, a_2, q_3)\cdots \in \delta^\omega \mid q_i \in F \text{ for some } i\}.$$

Furthermore, $\mathfrak{A}$ is a parity automaton, if

$$\mathrm{Acc} = \{(q_0, a_0, q_1)(q_1, a_1, q_2)(q_2, a_2, q_3)\cdots \in \delta^\omega \mid \limsup_{i\to\infty} \Omega(q_i) \text{ is even}\}$$

for some coloring $\Omega\colon Q \to \mathbb{N}$. To simplify our notation, define $\Omega(q, a, q') = \Omega(q)$. Finally, $\mathfrak{A}$ is a Muller automaton, if there is a family $\mathcal{F} \subseteq 2^Q$ of sets of states such that $\mathrm{Acc} = \{\rho \in \delta^\omega \mid \mathrm{Inf}(\rho) \in \mathcal{F}\}$, where $\mathrm{Inf}(\rho)$ is the set of states visited infinitely often by ρ.

2.2. Delay Games

A delay function is a mapping $f\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$, which is said to be constant if $f(i) = 1$ for all $i > 0$. A delay game $\Gamma_f(L)$ consists of a delay function f and a winning condition $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ for some alphabets Σ_I and Σ_O. Such a game is played in rounds $i = 0, 1, 2, \ldots$ as follows: in round i, first Player I picks a word $x_i \in \Sigma_I^{f(i)}$, then Player O picks a letter $y_i \in \Sigma_O$. Player O wins a play $(x_0, y_0)(x_1, y_1)(x_2, y_2)\cdots$ if the outcome $\binom{x_0 x_1 x_2 \cdots}{y_0 y_1 y_2 \cdots}$ is in L; otherwise, Player I wins.

A strategy for Player I in $\Gamma_f(L)$ is a mapping $\tau_I\colon \Sigma_O^* \to \Sigma_I^*$ satisfying $|\tau_I(w)| = f(|w|)$ while a strategy for Player O is a mapping $\tau_O\colon \Sigma_I^+ \to \Sigma_O$. A play $(x_0, y_0)(x_1, y_1)(x_2, y_2)\cdots$ is consistent with τ_I if $x_i = \tau_I(y_0 \cdots y_{i-1})$ for all i, and it is consistent with τ_O if $y_i = \tau_O(x_0 \cdots x_i)$ for all i. A strategy for Player $P \in \{I, O\}$ is winning, if every play that is consistent with the strategy is won by Player P.

An important special case are delay-free games, i.e., those with respect to the delay function f mapping every i to 1. In this case, we drop the subscript f and write $\Gamma(L)$ for the game with winning condition L. Such games are typically called Gale-Stewart games [3].

2.3. Finite-state Strategies in Gale-Stewart Games

A strategy for Player O in a Gale-Stewart game is still a mapping $\tau_O\colon \Sigma_I^+ \to \Sigma_O$. Such a strategy is said to be finite-state, if there is a deterministic finite transducer $\mathfrak{T}$ that implements τ_O in the following sense: $\mathfrak{T}$ is a tuple $(Q, \Sigma_I, q_I, \delta, \Sigma_O, \lambda)$ where Q is a finite set of states, Σ_I is the input alphabet, $q_I \in Q$ is the initial state, $\delta\colon Q \times \Sigma_I \to Q$ is the deterministic transition function, Σ_O is the output alphabet, and $\lambda\colon Q \to \Sigma_O$ is the output function. Let $\delta^*(q, x)$ denote the unique state that is reached by $\mathfrak{T}$ when processing $x \in \Sigma_I^*$ from $q \in Q$. Then, the strategy $\tau_{\mathfrak{T}}$ implemented by $\mathfrak{T}$ is defined as $\tau_{\mathfrak{T}}(x) = \lambda(\delta^*(q_I, x))$. We say that a strategy is finite-state, if it is implementable by some transducer. Slightly abusively, we identify finite-state strategies with transducers implementing them and talk about finite-state strategies with some number of states. Thus, we focus on the *state complexity* (e.g., the number of memory states necessary to implement a strategy) and ignore the other components of a transducer (which are anyway of polynomial size in $|Q|$, if we assume Σ_I and Σ_O to be fixed).

3. What is a Finite-state Strategy in a Delay Game?

Before we answer this question, we first ask what properties a finite-state strategy should have, i.e., what makes finite-state strategies in Gale-Stewart games useful and desirable? A strategy $\tau_O \colon \Sigma_I^+ \to \Sigma_O$ is in general an infinite object and does not necessarily have a finite representation. Furthermore, to execute such a strategy, one needs to store the whole sequence of moves made by Player I thus far: Unbounded memory is needed to execute it.

On the other hand, a finite-state strategy is finitely described by an automaton $\mathfrak{T}$ implementing it. To execute it, one only needs to store a single state of $\mathfrak{T}$ and access to the transition function δ and the output function λ of $\mathfrak{T}$. Assume the current state is q at the beginning of some round i (initialized with q_I before round 0). Then, Player I makes his move by picking some $a \in \Sigma_I$, which is processed by updating the memory state to $q' = \delta(q, a)$. Then, $\mathfrak{T}$ prescribes picking $\lambda(q') \in \Sigma_O$ and round i is completed.

Thus, there are two aspects that make finite-state strategies desirable: (1) the next move depends only on a finite amount of information about the history of the play, i.e., a state of the automaton, which is (2) easily updated. In particular, the necessary *machinery* of the strategy is encoded in the transition function and the output function.

Further, there is a generic framework to compute such strategies by reducing them to arena-based games.[3] As an example, consider a game $\Gamma(L(\mathfrak{A}))$ where $\mathfrak{A}$ is a parity automaton with set Q of states and transition function δ. We describe the construction of an arena-based parity game contested between Player I and Player O whose solution allows us to compute the desired strategies (formal details are presented in the appendix). The positions of Player I are states of $\mathfrak{A}$ while those of Player O are pairs (q, a) where $q \in Q$ and where a is an input letter. From a vertex q Player I can move to every state (q, a) for $a \in \Sigma_I$, from which Player O can move to every vertex $\delta(q, \binom{a}{b})$ for $b \in \Sigma_O$. Finally, Player O wins a play, if the run of $\mathfrak{A}$ constructed during the play is accepting. It is easy to see that the resulting game is a parity game with $|Q| \cdot (|\Sigma_I| + 1)$ vertices, and has the same winner as $\Gamma(L(\mathfrak{A}))$. The winner of the arena-based game has a positional[4] winning strategy [27, 28],

[3] Appendix A gives a short introduction to arena-based games.

[4] A strategy in an arena-based game is positional, if only depends on the last vertex of the play's history, not on the full history. A formal definition can be found in the appendix.

which can be computed in quasipolynomial time [29, 30, 31, 32]. Such a positional winning strategy can easily be turned into a finite-state winning strategy with $|Q| \cdot |\Sigma_I|$ states for Player O in the game $\Gamma(L(\mathfrak{A}))$, which is implemented by an automaton with state set $Q \times \Sigma_I$. This reduction can be generalized to arbitrary classes of Gale-Stewart games whose winning conditions are recognized by an ω-automaton with set Q of states: if Player O has a finite-state strategy with n' states in the arena-based game obtained by the construction described above, then Player O has a finite-state winning strategy with $|Q| \cdot |\Sigma_I| \cdot n'$ states for the original Gale-Stewart game. Such a strategy is obtained by solving an arena-based game with $|Q| \cdot (|\Sigma_I| + 1)$ vertices. Again, see the appendix for technical details.

3.1. Delay-oblivious Finite-state Strategies in Delay Games

So, what is a finite-state strategy in a delay game? In the following, we discuss this question for the case of delay games with respect to constant delay functions, which is the most important case. In particular, constant lookahead suffices for all ω-regular winning conditions [7], i.e, Player O wins with respect to an arbitrary delay function if, and only if, she wins with respect to a constant one. Similarly, constant lookahead suffices for many quantitative conditions like (parameterized) temporal logics [11] and parity conditions with costs [1]. For winning conditions given by parity automata, there is an exponential upper bound on the necessary constant lookahead. On the other hand, there are exponential lower bounds already for winning conditions specified by deterministic automata with reachability or safety acceptance [7] (which are subsumed by parity acceptance).

Technically, a strategy for Player O in a delay game is still a mapping $\tau_O \colon \Sigma_I^+ \to \Sigma_O$. Hence, the definition of finite-state strategies for Gale-Stewart games (see Subsection 2.3) is also applicable to delay games. With reasons that become apparent in the example succeeding the definition, we call such strategies *delay-oblivious*.

As a (cautionary) example, consider a delay game with winning condition $L_= = \{\binom{\alpha}{\alpha} \mid \alpha \in \{0,1\}^\omega\}$, i.e., Player O just has to copy Player I's moves, which she can do with respect to every delay function: Player O wins $\Gamma_f(L_=)$ for every f. However, a finite-state strategy has to remember the whole lookahead, i.e., those moves that Player I is ahead of Player O, in order to copy his moves. Thus, an automaton implementing a winning strategy for Player O in $\Gamma_f(L_=)$ needs at least $|\{0,1\}|^d$ states, if f is a constant delay function with $f(0) = d$. Thus, the memory requirements grow with the

size of the lookahead granted to Player O, i.e., lookahead is a burden, not an advantage. She even needs unbounded memory in the case of unbounded lookahead.

An advantage of this delay-oblivious definition is that finite-state strategies can be obtained by a trivial extension of the reduction presented for Gale-Stewart games above: now, states of Player I are from $Q \times \Sigma_I^{d-1}$ and those of Player O are from $Q \times \Sigma_I^d$. Player I can move from (q, w) to (q, wa) for $a \in \Sigma_I$ while Player O can move from (q, aw) to $(\delta(q, \binom{a}{b}), w)$ for $b \in \Sigma_O$. Intuitively, a state now additionally stores a queue of length $d-1$, which contains the lookahead granted to Player O. Coming back to the parity example, this approach yields a finite-state strategy with $|Q| \cdot |\Sigma_I|^d$ states. To obtain such a strategy, one has to solve a parity game with $|Q| \cdot (|\Sigma_I| + 1) \cdot |\Sigma_I|^{d-1}$ vertices, which is of doubly-exponential size in $|\mathfrak{A}|$, if d is close to the (tight) exponential upper bound. This can be done in doubly-exponential time, as it still has the same number of colors as the automaton $\mathfrak{A}$. Again, this reduction can be generalized to arbitrary classes of delay games with constant delay whose winning conditions are recognized by an ω-automaton with set Q of states: if Player O has a finite-state strategy with n' states in the arena-based game obtained by the construction, then Player O has a finite-state winning strategy with $|Q| \cdot |\Sigma_I|^d \cdot n'$ states for the delay game with constant lookahead of size d. In general, d factors exponentially into n', as n' is the memory size required to win a game with $\mathcal{O}(|\Sigma_I|^d)$ vertices. Also, to obtain the strategy for the delay game, one has to solve an arena-based game with $|Q| \cdot (|\Sigma_I| + 1) \cdot |\Sigma_I|^{d-1}$ vertices. Again, see the appendix for technical details.

3.2. Block Games

We show that one can do better by decoupling the history tracking and the handling of the lookahead, i.e., by using *delay-aware* finite-state strategies. In the delay-oblivious definition, we hardcode a queue into the arena-based game, which results in a blowup of the arena and therefore also in a blowup in the solution complexity and in the number of memory states for the arena-based game, which is turned into one for the delay game. To overcome this, we introduce a slight variation of delay games with respect to constant delay functions, so-called block games[5], present a notion of finite-state strategy in block games, and show how to transfer strategies between delay games and

[5]Holtmann, Kaiser, and Thomas already introduced a notion of block game in connection to delay games [6]. However, their notion differs from ours in several aspects. Most

block games. Then, we show how to solve block games and how to obtain finite-state strategies for them.

The motivation for introducing block games is to eliminate the queue containing the letters Player I is ahead of Player O, which is cumbersome to maintain, and causes the blowup in the case of games with winning condition $L_=$. Instead, in a block game, both players pick blocks of letters of a fixed length with Player I being one block ahead to account for the delay, i.e., Player I has to pick two blocks in round 0 and then one in every round, as does Player O in every round. This variant of delay games lies implicitly or explicitly at the foundation of all arguments establishing upper bounds on the necessary lookahead and at the foundations of all algorithms solving delay games [6, 7, 11, 12, 1]. Furthermore, we show how to transform a (winning) strategy for a delay game into a (winning) strategy for a block game and vice versa, i.e., Player O wins the delay game if, and only if, she wins the corresponding block game.[6]

Formally, the block game $\Gamma^d(L)$, where $d \in \mathbb{N} \setminus \{0\}$ is the block length and where $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ is the winning condition, is played in rounds as follows: in round 0, Player I picks two blocks $\overline{a_0}, \overline{a_1} \in \Sigma_I^d$, then Player O picks a block $\overline{b_0} \in \Sigma_O^d$. In round $i > 0$, Player I picks a block $\overline{a_{i+1}} \in \Sigma_I^d$, then Player O picks a block $\overline{b_i} \in \Sigma_O^d$. Player O wins the resulting play $\overline{a_0}\overline{a_1}\overline{b_0}\overline{a_2}\overline{b_1}\cdots$, if the outcome $\binom{a_0a_1a_2\cdots}{b_0b_1b_2\cdots}$ is in L.

A strategy for Player I in Γ is a map $\tau_I \colon (\Sigma_O^d)^* \to (\Sigma_I^d)^2 \cup \Sigma_I^d$ such that $\tau_I(\varepsilon) \in (\Sigma_I^d)^2$ and $\tau_I(\overline{b_0}\cdots\overline{b_i}) \in \Sigma_I^d$ for $i \geq 0$. A strategy for Player O is a map $\tau_O \colon (\Sigma_I^d)^* \to \Sigma_O^d$. A play $\overline{a_0}\overline{a_1}\overline{b_0}\overline{a_2}\overline{b_1}\cdots$ is consistent with τ_I, if $(\overline{a_0}, \overline{a_1}) = \tau_I(\varepsilon)$ and $\overline{a_i} = \tau_I(\overline{b_0}\cdots\overline{b_{i-2}})$ for every $i \geq 2$; it is consistent with τ_O if $\overline{b_i} = \tau_O(\overline{a_0}\cdots\overline{a_{i+1}})$ for every $i \geq 0$. Winning strategies and winning a block game are defined as for delay games.

The next lemma relates delay games with constant lookahead and block games: for a given winning condition, Player O wins a delay game with winning condition L (with respect to some delay function) if, and only if, she wins a block game with winning condition L (for some block size).

Lemma 1. *Let $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.*

importantly, in their definition, Player I determines the length of the blocks (within some bounds specified by f) while our block length is fixed and part of the rules of the game.

[6] Due to their importance and prevalence for solving delay games, one could even argue that the notion of block games is more suitable to model delay in infinite games.

1. *If Player O wins $\Gamma_f(L)$ for some constant delay function f, then she also wins $\Gamma^{f(0)}(L)$.*
2. *If Player O wins $\Gamma^d(L)$, then she also wins $\Gamma_f(L)$ for the constant delay function f with $f(0) = 2d$.*

Proof. 1.) Let $\tau_O \colon \Sigma_I^+ \to \Sigma_O$ be a winning strategy for Player O in $\Gamma_f(L)$ and fix $d = f(0)$. Now, define $\tau'_O \colon (\Sigma_I^d)^* \to (\Sigma_O)^d$ for Player O in $\Gamma^d(L)$ via $\tau'_O(\overline{a_0} \cdots \overline{a_i a_{i+1}}) = \beta(0) \cdots \beta(d-1)$ with $\beta(j) = \tau_O(\overline{a_0} \cdots \overline{a_i} \alpha(0)\alpha(1) \cdots \alpha(j-1))$ for $\overline{a_{i+1}} = \alpha(0)\alpha(1) \cdots \alpha(d-1)$.

A straightforward induction shows that for every play consistent with τ'_O there is a play consistent with τ_O that has the same outcome. Thus, as τ_O is a winning strategy, so is τ'_O.

2.) Now, let $\tau'_O \colon (\Sigma_I^d)^* \to (\Sigma_O)^d$ be a winning strategy for Player O in $\Gamma^d(L)$. We define $\tau_O \colon \Sigma_I^+ \to \Sigma_O$ for Player O in $\Gamma_f(L)$. To this end, let $x \in \Sigma_I^+$. By the choice of f, we obtain $|x| \geq f(0) = 2d$. Thus, we can decompose x into $x = \overline{a_0} \cdots \overline{a_i} x'$ such that $i \geq 1$, each $\overline{a_{i'}}$ is a block over Σ_I and $|x'| < d$. Now, let $\tau'_O(\overline{a_0} \cdots \overline{a_i}) = \beta(0) \cdots \beta(d-1)$. Then, we define $\tau_O(x) = \beta(|x'|)$.

Again, a straightforward induction shows that for every play consistent with τ_O there is a play consistent with τ'_O that has the same outcome. Thus, τ_O is a winning strategy. □

3.3. Delay-aware Finite-state Strategies in Block Games

After having proved the equivalence of block games and delay games w.r.t. constant delay, we now define *delay-aware* finite-state strategies for block games. Fix a block game $\Gamma^d(L)$ with $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$. A finite-state strategy for Player O in $\Gamma^d(L)$ is implemented by a transducer $\mathfrak{T} = (Q, \Sigma_I, q_I, \delta, \Sigma_O, \lambda)$ where Q, Σ_I, and q_I are defined as in Subsection 2.3. However, the transition function $\delta \colon Q \times \Sigma_I^d \to Q$ processes full input blocks and the output function $\lambda \colon Q \times \Sigma_I^d \times \Sigma_I^d \to \Sigma_O^d$ maps a state and a pair of input blocks to an output block. The strategy $\tau_{\mathfrak{T}}$ implemented by $\mathfrak{T}$ is defined as $\tau_{\mathfrak{T}}(\overline{a_0} \cdots \overline{a_i}) = \lambda(\delta^*(q_I, \overline{a_0} \cdots \overline{a_{i-2}}), \overline{a_{i-1}}, \overline{a_i})$ for $i \geq 1$. Here, $\delta^*(q, \overline{a_0} \cdots \overline{a_{i-2}})$ is the state reached by $\mathfrak{T}$ when processing $\overline{a_0} \cdots \overline{a_{i-2}}$ from q.

Example 1. Fix some $d > 0$. Player O has a trivial delay-aware finite-state winning strategy for $\Gamma^d(L_=)$ which is implemented by the transducer $(Q, \Sigma_I, q_I, \delta, \Sigma_O, \lambda)$ where $Q = \{q_I\}$, $\delta(q_I, \overline{a}) = q_I$ for every $q \in Q$ and every $\overline{a} \in \Sigma_I^d$, and where $\lambda(q, \overline{a_0}, \overline{a_1}) = \overline{a_0}$ for every $q \in Q$ and every $\overline{a_0}, \overline{a_1} \in \Sigma_I^d$.

Again, we identify delay-aware strategies with transducers implementing them and are interested in the number of states of the transducer. This captures the amount of information about a play's history that is differentiated in order to implement the strategy. Note that this ignores the representation of the transition and the output function. These are no longer "small" (in $|Q|$), as it is the case for transducers implementing strategies for Gale-Stewart games. When focussing on executing such strategies, these factors become relevant, but for our current purposes they are not: We have decoupled the history tracking from the lookahead-handling. The former is implemented by the automaton as usual while the latter is taken care of by the output function. In particular, the size of the automaton is (a-priori) independent of the block size. In Section 6, we revisit the issue of representing the transition and the output function succinctly, thereby addressing the issue of implementability.

In the next section, we present a very general approach to computing finite-state strategies for block games whose winning conditions are specified by automata with acceptance conditions that satisfy a certain aggregation property. For example, for block games with winning conditions given by deterministic parity automata, we obtain a strategy implemented by a transducer with exponentially many states, which can be obtained by solving a parity game of exponential size. In both aspects, this is an exponential improvement over the delay-oblivious variant for classical delay games.

To conclude the introduction of block games, we strengthen Lemma 1 to transfer finite-state strategies between delay games and block games.

Lemma 2. *Let $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.*

1. *If Player O has a delay-oblivious finite-state winning strategy for $\Gamma_f(L)$ with n states for some constant delay function f, then she also has a delay-aware finite-state winning strategy for $\Gamma^{f(0)}(L)$ with n states.*
2. *If Player O has a delay-aware finite-state winning strategy for $\Gamma^d(L)$ with n states, then she also has a delay-oblivious finite-state winning strategy for $\Gamma_f(L)$ with $n \cdot |\Sigma_I|^{2d}$ states for the constant delay function f with $f(0) = 2d$.*

Proof. It is straightforward to achieve the strategy transformations described in the proof of Lemma 1 by transforming transducers that implement finite-state strategies. □

The blowup in the direction from block games to delay games is in general unavoidable, as finite-state winning strategies for the game $\Gamma_f(L_=)$ need at least 2^d states to store the lookahead while winning strategies for the block game need only one state, independently of the block size.

4. Computing Finite-state Strategies for Block Games

The aim of this section is twofold. Our main aim is to compute finite-state strategies for block games (and, by extension, for delay games with constant lookahead). We do so by presenting a general framework for analyzing delay games with winning conditions specified by ω-automata whose acceptance conditions satisfy a certain aggregation property. The technical core is a reduction to a Gale-Stewart game, i.e., we remove the delay from the game. This framework yields upper bounds on the necessary (constant) lookahead to win a given game, but also allows to determine the winner and a finite-state winning strategy, if the resulting Gale-Stewart game can be effectively solved.

Slightly more formally, let $\mathfrak{A}$ be the automaton recognizing the winning condition of the block game. Then, the winning condition of the Gale-Stewart game constructed in the reduction is recognized by an automaton $\mathfrak{B}$ that can be derived from $\mathfrak{A}$. In particular, the acceptance condition of $\mathfrak{B}$ simulates the acceptance condition of $\mathfrak{A}$. Many types of acceptance conditions are preserved by the simulation, e.g., starting with a parity automaton $\mathfrak{A}$, we end up with a parity automaton $\mathfrak{B}$. Thus, the resulting Gale-Stewart game can be effectively solved.

Our second aim is to present a framework as general as possible to obtain upper bounds on the necessary lookahead and on the solution complexity for a wide range of winning conditions. In fact, our framework is a generalization and abstraction of techniques first developed for the case of ω-regular winning conditions [7], which were later generalized to other winning conditions [11, 12, 1]. Here, we cover all these results in a uniform way.

Let us begin by giving some intuition for the construction. The winning condition of the game is recognized by an automaton $\mathfrak{A}$. Thus, as usual, the exact input can be abstracted away, only the induced behavior in $\mathfrak{A}$ is relevant. Such a behavior is characterized by the state transformations induced by processing the input and by the effect on the acceptance condition triggered by processing it. For many acceptance conditions, this effect can

be aggregated, e.g., for parity conditions, one can decompose runs into non-empty pieces and then only consider the maximal colors of the pieces. For many quantitative winning conditions, one additionally needs bounds on the lengths of these pieces (cf. [12, 1]).

We first introduce aggregations and give some examples in Subsection 4.1 before we present the reduction to Gale-Stewart games using aggregations in Subsection 4.2

4.1. Aggregations

We begin by introducing two types of aggregations of varying strength. Fix an ω-automaton $\mathfrak{A} = (Q, \Sigma, q_I, \delta, \mathrm{Acc})$ and let $s\colon \delta^+ \to M$ for some finite set M. Given a decomposition $(\pi_i)_{i\in\mathbb{N}}$ of a run $\pi_0\pi_1\pi_2\cdots$ into non-empty pieces $\pi_i \in \delta^+$ we define $s((\pi_i)_{i\in\mathbb{N}}) = s(\pi_0)s(\pi_1)s(\pi_2)\cdots \in M^\omega$.

- We say that s is a strong aggregation (function) for $\mathfrak{A}$, if for all decompositions $(\pi_i)_{i\in\mathbb{N}}$ and $(\pi'_i)_{i\in\mathbb{N}}$ of any runs $\rho = \pi_0\pi_1\pi_2\cdots$ and $\rho' = \pi'_0\pi'_1\pi'_2\cdots$ with $\sup_i |\pi'_i| < \infty$ and $s((\pi_i)_{i\in\mathbb{N}}) = s((\pi'_i)_{i\in\mathbb{N}})$: $\rho \in \mathrm{Acc} \Rightarrow \rho' \in \mathrm{Acc}$.

- We say that s is a weak aggregation (function) for $\mathfrak{A}$, if for all decompositions $(\pi_i)_{i\in\mathbb{N}}$ and $(\pi'_i)_{i\in\mathbb{N}}$ of any runs $\rho = \pi_0\pi_1\pi_2\cdots$ and $\rho' = \pi'_0\pi'_1\pi'_2\cdots$ with $\sup_i |\pi_i| < \infty$, $\sup_i |\pi'_i| < \infty$, and $s((\pi_i)_{i\in\mathbb{N}}) = s((\pi'_i)_{i\in\mathbb{N}})$: $\rho \in \mathrm{Acc} \Rightarrow \rho' \in \mathrm{Acc}$.

Thus, in a strong aggregation, only the pieces π'_i of ρ' are of bounded length while in a weak aggregation both the pieces π_i of ρ and the pieces π'_i of ρ' are of bounded length.

Example 2.

- The function $s_{\mathrm{prty}}\colon \delta^+ \to \Omega(Q)$ defined as $s_{\mathrm{prty}}(t_0\cdots t_i) = \max_{0\le j\le i} \Omega(t_j)$ is a strong aggregation for a parity automaton $(Q, \Sigma, q_I, \delta, \mathrm{Acc})$ with coloring Ω.

- The function $s_{\mathrm{mllr}}\colon \delta^+ \to 2^Q$ defined as $s_{\mathrm{mllr}}((q_0, a_0, q_1)\cdots(q_n, a_n, q_{n+1})) = \{q_0, q_1, \ldots, q_n\}$ is a strong aggregation for a Muller automaton $(Q, \Sigma, q_I, \delta, \mathrm{Acc})$.

- The exponential time algorithm for delay games with winning conditions given by parity automata with costs, a quantitative generalization of parity automata, is based on a strong aggregation [1].

- The algorithm for delay games with winning conditions given by max automata [33], another quantitative automaton model, is based on a weak aggregation [12].

Due to symmetry, we can replace the implication $\rho \in \mathrm{Acc} \Rightarrow \rho' \in \mathrm{Acc}$ by an equivalence in the definition of a weak aggregation. Also, the notions are trivially hierarchical, i.e., every strong aggregation is also a weak one.

Let us briefly comment on the difference between strong and weak aggregations using the examples of parity automata with costs and max-automata: the acceptance condition of the former automata is a boundedness condition on some counters while the acceptance condition of the latter is a boolean combination of boundedness and unboundedness conditions on some counters. The aggregations for these acceptance conditions capture whether a piece of a run induces an increment of a counter or not, but abstract away the actual number of increments if it is non-zero. Now, consider the parity condition with costs, which requires to bound the counters. Assume the counters in some run $\pi_0\pi_1\pi_2\cdots$ are bounded and that we have pieces π_i' of bounded length having the same aggregation. Then, the increments in some piece π_i' have at least one corresponding increment in π_i. Thus, if a counter in $\pi_0'\pi_1'\pi_2'\cdots$ is unbounded, then it is also unbounded in $\pi_0\pi_1\pi_2\cdots$, which yields a contradiction. Hence, the implication $\pi_0\pi_1\pi_2\cdots \in \mathrm{Acc} \Rightarrow \pi_0'\pi_1'\pi_2'\cdots \in \mathrm{Acc}$ holds. For details, see [1]. On the other hand, to preserve boundedness and unboundedness properties, one needs to bound the length of the π_i' and the length of the π_i. Hence, there is only a weak aggregation for max-automata. Again, see [12] for details.

Given a weak aggregation s for $\mathfrak{A}$ with acceptance condition Acc, let

$$s(\mathrm{Acc}) = \{s((\pi_i)_{i\in\mathbb{N}}) \mid \pi_0\pi_1\pi_2\cdots \in \mathrm{Acc} \text{ is an accepting run of } \mathfrak{A} \text{ with } \sup_i |\pi_i| < \infty\}.$$

Next, we consider aggregations that are trackable by automata. A monitor for an automaton $\mathfrak{A}$ with transition function δ is a tuple $\mathfrak{M} = (M, \bot, \mathrm{upd})$ where M is a finite set of memory elements, $\bot \notin M$ is the empty memory element, and $\mathrm{upd}\colon M_\bot \times \delta \to M$ is an update function, where we use $M_\bot = M \cup \{\bot\}$. Note that the empty memory element $\bot$ is only used to initialize the memory, it is not in the image of upd. We say that $\mathfrak{M}$ computes the function $s_\mathfrak{M}\colon \delta^+ \to M$ defined by $s_\mathfrak{M}(t) = \mathrm{upd}(\bot, t)$ and $s_\mathfrak{M}(\pi \cdot t) = \mathrm{upd}(s_\mathfrak{M}(\pi), t)$ for $\pi \in \delta^+$ and $t \in \delta$.

Example 3. Recall Example 2. The strong aggregation s_{prty} for a parity automaton is computed by the monitor $(\Omega(Q), \bot, (c,t) \mapsto \max\{c, \Omega(t)\})$, where $\bot < c$ for every $c \in \Omega(Q)$.

Finally, we take the product of $\mathfrak{A}$ and the monitor $\mathfrak{M}$ for $\mathfrak{A}$, which simulates $\mathfrak{A}$ and simultaneously aggregates the acceptance condition. Formally, we define the product as $\mathfrak{A} \times \mathfrak{M} = (Q \times M_\bot, (q_I, \bot), \Sigma, \delta', \emptyset)$ where $\delta'((q,m),a) = (q', \mathrm{upd}(m,(q,a,q')))$ for $q' = \delta(q,a)$. Note that $\mathfrak{A} \times \mathfrak{M}$ has an empty set of accepting runs, as these are irrelevant to us.

4.2. Removing Delay via Aggregations

Consider a play prefix in a delay game $\Gamma_f(L(\mathfrak{A}))$: Player I has produced a sequence $\alpha(0)\cdots\alpha(i)$ of letters while Player O has produced $\beta(0)\cdots\beta(i')$ with, in general, $i' < i$. Now, she has to determine $\beta(i'+1)$. The automaton $\mathfrak{A} \times \mathfrak{M}$ can process the joint sequence $\binom{\alpha(0)\cdots\alpha(i')}{\beta(0)\cdots\beta(i')}$, but not the sequence $\alpha(i'+1)\cdots\alpha(i)$, as Player O has not yet picked the letters $\beta(i'+1)\cdots\beta(i)$. However, one can determine which states are reachable by some completion $\binom{\alpha(i'+1)\cdots\alpha(i)}{\beta(i'+1)\cdots\beta(i)}$ by projecting away Σ_O from $\mathfrak{A} \times \mathfrak{M}$.

Thus, from now on assume $\Sigma = \Sigma_I \times \Sigma_O$ and define $\delta_P \colon 2^{Q \times M_\bot} \times \Sigma_I \to 2^{Q \times M}$ (P for power set) via

$$\delta_P(S,a) = \left\{ \delta'\left((q,m), \binom{a}{b}\right) \,\middle|\, (q,m) \in S \text{ and } b \in \Sigma_O \right\}.$$

Intuitively, δ_P is obtained as follows: take $\mathfrak{A} \times \mathfrak{M}$, project away Σ_O, and apply the power set construction (while discarding the anyway empty acceptance condition). Then, δ_P is the transition function of the resulting deterministic automaton. As usual, we extend δ_P to $\delta_P^+ \colon 2^{Q \times M_\bot} \times \Sigma_I^+ \to 2^{Q \times M}$ via $\delta_P^+(S,a) = \delta_P(S,a)$ and $\delta_P^+(S,wa) = \delta_P(\delta_P^+(S,w),a)$.

Given states q and q' of $\mathfrak{A}$, a memory state m, and a word $w \in \Sigma_I^+$, we call a word $w' \in \Sigma_O^{|w|}$ a (q,q',m)-completion of w, if the run π of $\mathfrak{A}$ processing $\binom{w}{w'}$ starting from q ends in q' and satisfies $s_{\mathfrak{M}}(\pi) = m$.

Remark 1. The following are equivalent for $q \in Q$ and $w \in \Sigma_I^+$:

1. $(q', m') \in \delta_P^+(\{(q, \bot)\}, w)$.
2. There is a (q, q', m')-completion of w.

We use this property to define an equivalence relation formalizing the idea that words having the same behavior in $\mathfrak{A} \times \mathfrak{M}$ do not need to be distinguished. To this end, to every $w \in \Sigma_I^+$ we assign the transition summary $r_w \colon Q \to 2^{Q \times M}$ defined via $r_w(q) = \delta_P^+(\{(q, \bot)\}, w)$. Having the same transition summary is a finite equivalence relation $\equiv$ over Σ_I^+ whose index is bounded by $2^{|Q|^2|M|}$. For an $\equiv$-class $S = [w]_\equiv$ define $r_S = r_w$, which is independent of representatives. Let R be the set of infinite $\equiv$-classes.

Now, we define a Gale-Stewart game in which Player I determines an infinite sequence of equivalence classes from R. By picking representatives, this induces a word $\alpha \in \Sigma_I^\omega$. Player O picks states (q_i, m_i) such that the m_i aggregate a run of $\mathfrak{A}$ on some completion $\binom{\alpha}{\beta}$ of α. Player O wins if the m_i imply that the run of $\mathfrak{A}$ on $\binom{\alpha}{\beta}$ is accepting. To account for the delay, Player I is always one move ahead, which is achieved by adding a dummy move for Player O in round 0.

Formally, in round 0, Player I picks an $\equiv$-class $S_0 \in R$ and Player O has to pick $(q_0, m_0) = (q_I, \bot)$. In round $i > 0$, first Player I picks an $\equiv$-class $S_i \in R$, then Player O picks a state $(q_i, m_i) \in r_{S_{i-1}}(q_{i-1})$ of the product automaton. Player O wins the resulting play $S_0(q_0, m_0)S_1(q_1, m_1)S_2(q_2, m_2) \cdots$ if $m_1 m_2 m_3 \cdots \in s_{\mathfrak{M}}(\mathrm{Acc})$ (note that m_0 is ignored). The notions of (finite-state and winning) strategies are inherited from Gale-Stewart games, as this game is indeed such a game $\Gamma(L(\mathfrak{B}))$ for some automaton $\mathfrak{B}$ of size $|R| \cdot |Q| \cdot |M|$ which can be derived from $\mathfrak{A}$ and $\mathfrak{M}$ as follows:

Let $\mathfrak{A} = (Q, \Sigma_I \times \Sigma_O, q_I, \delta, \mathrm{Acc})$ and $\mathfrak{M} = (M, \bot, \mathrm{upd})$ be given. We define $\mathfrak{B} = (R \times Q \times M_\bot, R \times (Q \times M), (S_I, q_I, m_I), \delta', \mathrm{Acc}')$ for some arbitrary $S_I \in R$, some arbitrary $m_I \in M$, $\delta'((S, q, m), \binom{S'}{(q', m')}) = (S', q', m')$, and $(S_0, q_0, m_0)(S_1, q_1, m_1)(S_2, q_2, m_2) \cdots \in \mathrm{Acc}'$ if, and only if,

- $(q_0, m_0) = (q_I, \bot)$,
- $(q_i, m_i) \in r_{S_{i-1}}(q_{i-1})$ for all $i > 0$, and
- $m_1 m_2 m_3 \cdots \in s_{\mathfrak{M}}(\mathrm{Acc})$.

It is straightforward to prove that $\mathfrak{B}$ has the desired properties.

Note that, due to our very general definition of acceptance conditions, we are able to express the local consistency requirement "$(q_i, m_i) \in r_{S_{i-1}}(q_{i-1})$" using the acceptance condition. For less general acceptance modes, e.g., parity, one has to check this property using the state space of the automaton,

which leads to a polynomial blowup, as one has to store each S_{i-1} for one transition.

Theorem 1. *Let $\mathfrak{A}$ be an ω-automaton and let $\mathfrak{M}$ be a monitor for $\mathfrak{A}$ such that $s_{\mathfrak{M}}$ is a strong aggregation for $\mathfrak{A}$, let $\mathfrak{B}$ be constructed as above, and define $d = 2^{|Q|^2 \cdot |M_\perp|}$.*

1. *If Player O wins $\Gamma_f(L(\mathfrak{A}))$ for some delay function f, then she also wins $\Gamma(L(\mathfrak{B}))$.*
2. *If Player O wins $\Gamma(L(\mathfrak{B}))$, then she also wins the block game $\Gamma^d(L(\mathfrak{A}))$. Moreover, if she has a finite-state winning strategy for $\Gamma(L(\mathfrak{B}))$ with n states, then she has a delay-aware finite-state winning strategy for $\Gamma^d(L(\mathfrak{A}))$ with n states.*

Before we prove these results, we need to establish a closure property of the sets $s(\mathrm{Acc})$ in case s is a strong aggregation for an ω-automaton with acceptance condition Acc. Recall that we defined

$$s(\mathrm{Acc}) = \{s((\pi_i)_{i\in\mathbb{N}}) \mid \pi_0\pi_1\pi_2\cdots \in \mathrm{Acc} \text{ is an accepting run of } \mathfrak{A} \text{ with } \sup_i |\pi_i| < \infty\},$$

i.e., $s(\mathrm{Acc})$ only contains the aggregations of decompositions into pieces of bounded length. However, if s is strong, then this restriction is not essential: the π_i in the following lemma are not required to be of bounded length.

Lemma 3. *Let s be a strong aggregation for an ω-automaton $\mathfrak{A}$ with acceptance condition* Acc *and let $\pi_0\pi_1\pi_2\cdots \in \mathrm{Acc}$. Then, $s((\pi_i)_{i\in\mathbb{N}}) \in s(\mathrm{Acc})$.*

Proof. The s-profile of a finite run π is the tuple (q, m, q') where q is the state π starts in, $m = s(\pi)$, and q' is the state π ends in. Having the same s-profile is an equivalence relation over finite runs of finite index. For each equivalence class S of this relation, let $\mathrm{rep}(S)$ be an arbitrary, but fixed, element of S. For notational convenience, define $\mathrm{rep}(\pi) = \mathrm{rep}(S)$ for the unique equivalence class S with $\pi \in S$.

Now, consider the sequence $\mathrm{rep}(\pi_0)\mathrm{rep}(\pi_1)\mathrm{rep}(\pi_2)\cdots$. By construction, it is also a run of $\mathfrak{A}$ and we have $s((\pi_i)_{i\in\mathbb{N}}) = s((\mathrm{rep}(\pi_i))_{i\in\mathbb{N}})$. As the $\mathrm{rep}(\pi_i)$ are of bounded length (after all, there are only finitely many representatives), s being a strong aggregation yields $\mathrm{rep}(\pi_0)\mathrm{rep}(\pi_1)\mathrm{rep}(\pi_2)\cdots \in \mathrm{Acc}$. Hence, $s((\pi_i)_{i\in\mathbb{N}}) = s((\mathrm{rep}(\pi_i))_{i\in\mathbb{N}}) \in s(\mathrm{Acc})$. □

Also, we need some basic properties of equivalence classes $S \in R$. They follow from the fact that every equivalence class S, which is a language of finite words, is recognized by a deterministic finite automaton of size d (as defined above) obtained using the state set $(2^{Q \times M_\perp})^Q$ to simulate δ_P^+ starting from the states of the form $\{(q, \perp)\}$.

Remark 2.

1. $[w]_\equiv \in R$ for every w of length at least d.
2. Let $S \in R$. For every n, S contains a word w of length $n \leq |w| \leq n+d$.

Now, we are ready to prove Theorem 1.

Proof. The argument is a further generalization of similar constructions for parity automata (with or without costs) and max-automata (cp. [7, 12, 1]).

1.) Let $\tau_O^f \colon \Sigma_I^+ \to \Sigma_O$ be a winning strategy for Player O in $\Gamma_f(L(\mathfrak{A}))$ for some fixed f. For the sake of readability, we denote $\Gamma_f(L(\mathfrak{A}))$ by Γ_f and $\Gamma(L(\mathfrak{B}))$ by Γ. We describe how to simulate a play in Γ by a play in Γ_f to transform τ_O^f into a winning strategy τ_O for Player O in Γ.

To this end, let Player I pick $S_0 \in R$ in Γ, which has to be answered by Player O by picking $(q_0, m_0) = (q_I, \perp)$. Thus, we define $\tau_O(S_0) = (q_0, m_0)$. Next, Player I picks some $S_1 \in R$.

To simulate this, pick some $x_0 \in S_0$ satisfying $|x_0| \geq f(0)$, which exists due to S_0 being infinite by virtue of being in R. Similarly, we pick some $x_1 \in S_1$ satisfying $|x_0 x_1| \geq \sum_{j=0}^{|x_0|-1} f(j)$, which again exists due to S_1 being infinite.

Now, assume Player I starts a play by picking the letters of the prefix of $x_0 x_1$ of length $\sum_{j=0}^{|x_0|-1} f(j)$ during the first $|x_0|$ rounds of Γ_f. By the choice of $|x_1|$, $x_0 x_1$ is long enough to do so. Let y_0 be the answer of Player O according to τ_O during these $|x_0|$ rounds, i.e., $|y_0| = |x_0|$.

Thus, we are in the following situation for $i = 1$:

- In Γ, the players have produced the play prefix $S_0(q_0, m_0) \cdots S_{i-1}(q_{i-1}, m_{i-1}) S_i$.
- In Γ_f, Player I has picked a prefix of $x_0 \cdots x_i$ while Player O has picked $y_0 \cdots y_{i-1}$ according to τ_O^f such that $|y_0 \cdots y_{i-1}| = |x_0 \cdots x_{i-1}|$. Furthermore, we have $[x_j]_\equiv = S_j$ for every $j \leq i$.

Now, let $i > 0$ be arbitrary. Let q_i be the state reached by $\mathfrak{A}$ when processing $\binom{x_{i-1}}{y_{i-1}}$ when starting in q_{i-1}, let π_{i-1} be the corresponding run, and define

$m_i = s_{\mathfrak{M}}(\pi_{i-1})$. Then, by definition of r_{S_i} and by Remark 1, $\tau_O(S_0 \cdots S_i) = (q_i, m_i)$ is a legal move in Γ, which is answered by Player I picking some $S_{i+1} \in R$. Again, we pick some $x_{i+1} \in S_{i+1}$ such that $|x_0 \cdots x_{i+1}| \geq \sum_{j=0}^{|x_0 \cdots x_i|-1} f(j)$ and consider the play prefix of Γ_f where Player I starts by picking the letters of the prefix of $x_0 \cdots x_{i+1}$ of length $\sum_{j=0}^{|x_0 \cdots x_i|-1} f(j)$ during the first $|x_0 \cdots x_i|$ rounds, which is a continuation of the previously defined one. Player O answers the letters of x_i by some y_i of the same length. Thus, we are in the situation above for $i+1$, which concludes the inductive definition of τ_O.

To conclude, we show that τ_O is indeed winning for Player O in Γ. So, let $w = S_0(q_0, m_0)S_1(q_1, m_1)S_2(q_2, m_2) \cdots$ be a play consistent with τ_O and let $w^f = \binom{x_0}{y_0}\binom{x_1}{y_1}\binom{x_2}{y_2} \cdots$ be the outcome of the simulated play of Γ_f as described above. By construction, w^f is in $L(\mathfrak{A})$, as the simulated play is consistent with the winning strategy τ_O^f.

Let π_i be defined as above, i.e., $\pi_0\pi_1\pi_2 \cdots$ is the run of $\mathfrak{A}$ on w^f and therefore accepting. By construction, we have $s_{\mathfrak{M}}(\pi_i) = m_{i+1}$. Applying Lemma 3 yields $m_1m_2m_3 \cdots = s_{\mathfrak{M}}((\pi_i)_{i\in\mathbb{N}}) \in s_{\mathfrak{M}}(\mathrm{Acc})$. Thus, w is indeed winning for Player O.

2.) Let τ_O be a winning strategy for Player O in $\Gamma(L(\mathfrak{B}))$. Again, for the sake of readability, we denote $\Gamma(L(\mathfrak{B}))$ by Γ and $\Gamma^d(L(\mathfrak{A}))$ by Γ^d. As before, we simulate a play in Γ^d by a play in Γ to transform τ_O into a winning strategy τ_O^d for Player O in Γ^d. In the following proof, all blocks $\overline{a_i}$ are in Σ_I^d and all $\overline{b_i}$ are in Σ_O^d.

Thus, let Player I pick $\overline{a_0}$ and $\overline{a_1}$ during the first round in Γ^d and define $S_0 = [\overline{a_0}]_{\equiv}$, $(q_0, m_0) = \tau_O(S_0)$, $S_1 = [\overline{a_1}]_{\equiv}$, and $(q_1, m_1) = \tau_O(S_0S_1)$.

Now, we are in the following situation for $i = 1$.

- In Γ^d Player I has picked $\overline{a_0} \cdots \overline{a_i}$ and Player O has picked $\overline{b_0} \cdots \overline{b_{i-2}}$ (which is empty for $i = 1$).
- In Γ, we have constructed the play prefix $S_0(q_0, m_0) \cdots S_{i-1}(q_{i-1}, m_{i-1})S_i(q_i, m_i)$ that is consistent with τ_O and satisfies $S_j = [\overline{a_j}]_{\equiv}$ for every $j \leq i$.

Now, let $i > 0$ be arbitrary. By definition of Γ, we have $(q_i, m_i) \in r_{\overline{a_{i-1}}}(q_{i-1})$. Thus, by definition of $r_{\overline{a_{i-1}}}$ and Remark 1 there is a $\overline{b_{i-1}}$ such that the run π_{i-1} of $\mathfrak{A}$ processing $\binom{\overline{a_{i-1}}}{\overline{b_{i-1}}}$ from q_{i-1} ends in q_i and satisfies $s_{\mathfrak{M}}(\pi_{i-1}) = m_i$. We define $\tau_O^d(\overline{a_0} \cdots \overline{a_i}) = \overline{b_{i-1}}$. This move is answered by

Player I picking some block a_{i+1}, which again induces $S_{i+1} = [\overline{a_{i+1}}]_{\equiv}$. Applying τ_O yields $(q_{i+1}, m_{i+1}) = \tau_O(S_0 \cdots S_{i+1})$. Thus, we are in the situation described above for $i+1$, which completes the inductive definition of τ_O^d.

Note that if τ_O is implemented by a transducer $\mathfrak{T}$ with n states, then τ_O^d can easily be implemented by an automaton with n states, which is obtained from $\mathfrak{T}$ as follows: we use the same set of states so that processing $\overline{a_0} \cdots \overline{a_{i-2}}$ leads to the state reached when processing $[\overline{a_0}]_{\equiv} \cdots [\overline{a_{i-2}}]_{\equiv}$, call it q. Now, assume we have two additional blocks $\overline{a_{i-1}}$ and $\overline{a_i}$ and have to compute the block $\overline{b_{i-1}} = \tau_O^d(\overline{a_0} \cdots \overline{a_i})$ as defined above. This block only depends on the state q of the automaton implementing the strategy, on the states q_{i-1} and q_i of $\mathfrak{A}$, on m_i, and on $\overline{a_{i-1}}$. All this information can be computed from q and the moves $[\overline{a_{i-1}}]_{\equiv}$ and $[\overline{a_i}]_{\equiv}$ of Player I in the simulating play.

It remains to show that τ_O^d is indeed a winning strategy for Player O in Γ^d. To this end, let $w^d = \overline{a_0 a_1} \overline{b_0} \overline{a_2} \overline{b_1} \cdots$ a play that is consistent with τ_O^d. Furthermore, let $S_0(q_0, m_0) S_1(q_1, m_1) S_2(q_2, m_2) \cdots$ be the simulated play in Γ constructed as described above, which is consistent with τ_O. Therefore, it is winning for Player O, i.e., $m_1 m_2 m_3 \cdots \in s_{\mathfrak{M}}(\mathrm{Acc})$.

Let the finite runs π_i be defined as above, i.e., $\pi_0 \pi_1 \pi_2 \cdots$ is the run of $\mathfrak{A}$ on w^d and the part π_i processes $\binom{\overline{a_i}}{\overline{b_i}}$. Thus, the length of each π_i is equal to d. Furthermore, we have $s_{\mathfrak{M}}(\pi_i) = m_{i+1}$ for every i. From $s_{\mathfrak{M}}((\pi_i)_{i \in \mathbb{N}}) = m_1 m_2 m_3 \cdots \in s_{\mathfrak{M}}(\mathrm{Acc})$ and $s_{\mathfrak{M}}$ being a weak aggregation (as it is strong), we conclude that $\pi_0 \pi_1 \pi_2 \cdots$ is accepting. Hence, $w^d \in L(\mathfrak{A})$, i.e., Player O wins the play. □

By applying both implications and Item 2 of Lemma 1, we obtain upper bounds on the complexity of determining for a given $\mathfrak{A}$ whether Player O wins $\Gamma_f(L(\mathfrak{A}))$ for some f and on the constant lookahead necessary to do so.

Corollary 1. *Let $\mathfrak{A}$, $\mathfrak{M}$, $\mathfrak{B}$, and d be as in Theorem 1. Then, the following are equivalent:*

1. *Player O wins $\Gamma_f(L(\mathfrak{A}))$ for some delay function f.*
2. *Player O wins $\Gamma_f(L(\mathfrak{A}))$ for the constant delay function f with $f(0) = 2d$.*
3. *Player O wins $\Gamma(L(\mathfrak{B}))$.*

Thus, determining whether, given $\mathfrak{A}$, Player O wins $\Gamma_f(L(\mathfrak{A}))$ for some f is achieved by determining the winner of the Gale-Stewart game $\Gamma(L(\mathfrak{B}))$ and, independently, we obtain an exponential upper bound on the necessary constant lookahead (in $|Q| \cdot |M|$).

Example 4. Continuing our example for the parity acceptance condition, we obtain the exponential upper bound $2^{|Q|^2 \cdot |\Omega(Q)|+2}$ on the constant lookahead necessary to win the delay game and an exponential-time algorithm for determining the winner, as $\mathfrak{B}$ has exponentially many states, but the same number of colors as $\mathfrak{A}$. Both upper bounds are tight [7].

In case there is no strong aggregation for $\mathfrak{A}$, but only a weak one, one can show that finite-state strategies exist, if Player O wins with respect to some constant delay function at all.

Theorem 2. *Let $\mathfrak{A}$ be an ω-automaton and let $\mathfrak{M}$ be a monitor for $\mathfrak{A}$ such that $s_{\mathfrak{M}}$ is a weak aggregation for $\mathfrak{A}$, let $\mathfrak{B}$ be constructed as above, and define $d = 2^{|Q|^2 \cdot |M_\perp|}$.*

1. *If Player O wins $\Gamma_f(L(\mathfrak{A}))$ for some constant delay function f, then she also wins $\Gamma(L(\mathfrak{B}))$.*
2. *If Player O wins $\Gamma(L(\mathfrak{B}))$, then she also wins the block game $\Gamma^d(L(\mathfrak{A}))$. Moreover, if she has a finite-state winning strategy for $\Gamma(L(\mathfrak{B}))$ with n states, then she has a delay-aware finite-state winning strategy for $\Gamma^d(L(\mathfrak{A}))$ with n states.*

Proof. The second implication is the same as the second one in Theorem 1, in whose proof we only required s to be a weak aggregation, which is the setting here. Hence, we only have to consider the first implication.

To this end, we construct a strategy τ_O for Player O in $\Gamma(L(\mathfrak{B}))$ from a winning strategy τ_O^f for Player O in $\Gamma_f(L(\mathfrak{A}))$ as described in the proof of Item 1 of Theorem 1. The only difference is that here we can ensure that the length of the x_i is bounded, as f is constant. This allows us to replace the invocation of Lemma 3 and directly apply the definition of $s_{\mathfrak{M}}(\mathrm{Acc})$ to show that the plays consistent with τ_O are winning for Player O. □

Again, we obtain upper bounds on the solution complexity (here, with respect to constant delay functions) and on the necessary constant lookahead.

Corollary 2. *Let $\mathfrak{A}$, $\mathfrak{M}$, $\mathfrak{B}$, and d be as in Theorem 2. Then, the following are equivalent:*

1. *Player O wins $\Gamma_f(L(\mathfrak{A}))$ for some constant delay function f.*
2. *Player O wins $\Gamma_f(L(\mathfrak{A}))$ for the constant delay function f with $f(0) = 2d$.*
3. *Player O wins $\Gamma(L(\mathfrak{B}))$.*

5. Discussion

Let us compare the two approaches presented in the previous section with three use cases: delay games whose winning conditions are given by deterministic parity automata, by deterministic Muller automata, and by LTL formulas. All formalisms only define ω-regular languages, but vary in their succinctness.

The following facts about arena-based games will be useful for the comparison:

- The winner of an arena-based parity game has a positional winning strategy [27, 28], i.e., a finite-state strategy with a single state.
- The winner of an arena-based Muller game has a finite-state strategy with $n!$ states [34], where n is the number of vertices of the arena.
- The winner of an arena-based LTL game has a finite-state strategy with $2^{2^{\mathcal{O}(|\varphi|)}}$ states [35], where φ is the formula specifying the winning condition.

Also, we need the following bounds on the necessary lookahead in delay games:

- In delay games whose winning conditions are given by deterministic parity automata, exponential (in the size of the automata) constant lookahead is both sufficient and in general necessary [7].
- In delay games whose winning conditions are given by deterministic Muller automata, doubly-exponential (in the size of the automata) constant lookahead is sufficient. This follows from the transformation of deterministic Muller automata into deterministic parity automata of exponential size (see, e.g., [36]). However, the best lower bound is the exponential one for parity automata, which are also Muller automata.
- In delay games whose winning conditions are given by LTL formulas, triply-exponential (in the size of the formula) constant lookahead is both sufficient and in general necessary [11].

Using these facts, we obtain the following complexity results for finite-state strategies: Figure 1 shows the upper bounds on the number of states of delay-oblivious finite-state strategies for delay games and on the number of

states of delay-aware finite-state strategies for block games and upper bounds on the complexity of determining such strategies. In all three cases, the former strategies are at least exponentially larger and at least exponentially harder to compute. This illustrates the advantage of decoupling tracking the history from managing the lookahead.

	parity	Muller	LTL
delay-oblivious	doubly-exp.	quadruply-exp.	quadruply-exp.
delay-aware	exp.	doubly-exp.	triply-exp.

Figure 1: Memory size for delay-oblivious strategies (for delay games) and delay-aware finite-state strategies (for block games), measured in the size of the representation of the winning condition. For the sake of readability, we only present the orders of magnitude, but not exact values.

Finally, let us compare our approach to that of Salzmann. Fix a delay game $\Gamma_f(L(\mathfrak{A}))$ and assume Player I has picked $\alpha(0)\cdots\alpha(i)$ while Player O has picked $\beta(0)\cdots\beta(i')$ with $i' < i$. His strategies are similar to our delay-aware ones for block games. The main technical difference is that his strategies have access to the state reached by $\mathfrak{A}$ when processing $\binom{\alpha(0)\cdots\alpha(i')}{\beta(0)\cdots\beta(i')}$. Thus, his strategies explicitly depend on the specification automaton $\mathfrak{A}$ while ours are independent of it. In general, his strategies are therefore smaller than ours, as our transducers have to simulate $\mathfrak{A}$ if they need access to the current state. On the other hand, our aggregation-based framework is more general and readily applicable to quantitative winning conditions as well, while he only presents results for selected qualitative conditions like parity, weak parity, and Muller.

6. Succinctly Implementing Finite-state Strategies for Block Games

In the previous section, we have shown how to compute delay-aware finite-state strategies for block games via a reduction to Gale-Stewart games. The transducers implementing these strategies process blocks of letters, i.e., the domains of the transition function and of the output function are (roughly) of size $|\Sigma_I|^d$ and $|\Sigma_I|^{2d}$, where d is the block size of the block game. It is known that even for very simple winning conditions, an exponential d is necessary (measured in the size of the automaton $\mathfrak{A}$ recognizing the winning condition). In this case, the representation of these transducers is at least of

doubly-exponential size in $|\mathfrak{A}|$, independently of the number of states of the transducer.

In this section, we propose a succinct notion of transducers implementing delay-aware strategies which can be significantly smaller, e.g., of constant size for the winning condition $L_=$ introduced in Section 3, which requires Player O to copy the moves of Player I. Here, the size of the domains of the transition function and of the output function grows exponentially with the block size d, although the transition function and the output function of a transducer implementing a winning strategy are trivial.

Intuitively, to obtain succinct transducers implementing strategies in block games, we implement the transition function and the output function by transducers. As already alluded to, we present examples (see Examples 5 and 6) in which this representation is much smaller than the explicit representation. Furthermore, we give an upper bound on the size of such succinct transducers which is asymptotically equal to the true representation size of explicit transducers in Subsection 6.2. Thus, succinct transducers are never larger than explicit ones. However, we also present an example where they cannot be smaller than explicit ones in Subsection 6.3. Finally, we discuss the relation between block sizes and sizes of succinct transducers in Subsection 6.4.

6.1. Succinct Transducers

Formally, for a block game $\Gamma^d(L)$ with $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$, we implement a finite-state strategy for Player O in $\Gamma^d(L)$ by a transducer $\mathfrak{T} = (Q, \Sigma_I, q_I, \Delta, \Sigma_O, \Lambda)$, where Q, Σ_I, q_I, and Σ_O are defined as before in Subsection 3.3. However, the transition function and the output function are now succinctly represented by transducers Δ and Λ which we define below, respectively. From now on, we refer to this type of transducer as *succinct transducer* and to the type introduced in Subsection 3.3 as explicit transducer. Regarding succinct transducers, we speak of "master states" to refer to Q, and we speak of "transition slave" and "output slave" to refer to Δ and Λ, respectively.

The transition slave is a tuple $\Delta = (Q_\Delta, \Sigma_I, q_I^\Delta, \delta, Q, \lambda)$, where Q_Δ is a finite set of states, $q_I^\Delta \colon Q \to Q_\Delta$ is a function returning an initial state, $\delta \colon Q_\Delta \times \Sigma_I \to Q_\Delta$ is the transition function, and $\lambda \colon Q_\Delta \to Q$ is the output function. We say that the transition slave computes the function $\Delta \colon Q \times \Sigma_I^* \to Q$ defined by $\Delta(q, x) = \lambda(\delta^*(q_I^\Delta(q), x))$, where $\delta^*(q_I^\Delta(q), x)$ is the state of Δ reached by processing x from the state $q_I^\Delta(q)$ of Δ. The size of Δ is defined as $|\Delta| = |Q_\Delta|$.

The output slave is a tuple $\Lambda = (Q_\Lambda, \Sigma_I, q_I^\Lambda, E, \Sigma_O)$, where Q_Λ is a finite set of states, Σ_I is the input alphabet, $q_I^\Lambda \colon Q \to Q_\Lambda$ is a function returning an initial state, and $E \colon Q_\Lambda \times (\Sigma_I \cup \{\$\}) \to \Sigma_O^* \times Q_\Lambda$ is the deterministic transition function conveniently treated as a relation. Here, \$ is a fresh symbol that is used to separate input blocks.

A finite run π of Λ is a sequence

$$\pi = (q_0, a_0, b_0, q_1)(q_1, a_1, b_1, q_2) \cdots (q_{i-1}, a_{i-1}, b_{i-1}, q_i) \in E^+.$$

We say that π starts in q_0, ends in q_i, its processed input is $in(\pi) = a_0 \cdots a_{i-1} \in (\Sigma_I \cup \{\$\})^+$, and its produced output is $out(\pi) = b_0 \cdots b_{i-1} \in \Sigma_O^*$. We say that the output slave computes the function $\Lambda \colon Q \times \Sigma_I^+ \times \Sigma_I^+ \to \Sigma_O^*$ defined by $\Lambda(q, x_1, x_2) = out(\pi)$, where π is the unique run that starts in $q_I^\Lambda(q)$ with $in(\pi) = x_1\$x_2\$$. The size of Λ is defined as $|\Lambda| = |Q_\Lambda| + \ell$, where ℓ is the length of the longest output in E, that is, $\max\{|v| \mid (p, u, v, q) \in E\}$.

Clearly, if additionally $\Lambda(q, \overline{a_0}, \overline{a_1}) \in \Sigma_O^d$ for every $q \in Q$ and every $\overline{a_0}, \overline{a_1} \in \Sigma_I^d$, then the succinct transducer $\mathfrak{T}$ implements a strategy $\tau_\mathfrak{T}$ as before, namely $\tau_\mathfrak{T}(\overline{a_0} \cdots \overline{a_i}) = \Lambda(\Delta^*(q_I, \overline{a_0} \cdots \overline{a_{i-2}}), \overline{a_{i-1}}, \overline{a_i})$ for $i \geq 1$. The size of $\mathfrak{T}$ is defined as $|\mathfrak{T}| = |Q| + |\Delta| + |\Lambda|$.

We illustrate these definitions with two examples. In the first one, we substantiate our above claim that a succinct transducer of constant size implements a winning strategy for Player O in $\Gamma^d(L_=)$, independently of d. This is in sharp contrast to explicit transducers implementing winning strategies, whose transition and output function have exponentially-sized domains in d.

Example 5. Consider the winning condition $L_=$ as introduced in Section 3. Obviously, Player O can win the block game $\Gamma^d(L_=)$ by copying the moves of Player I for every block size d. A succinct transducer implementing a winning strategy for Player O can be defined independently of d. One master state, one state for the transition slave, and one state for the output slave suffice; the output slave just copies the input until the first \$ occurs and ignores the remaining input.

One obvious weakness of the previous example is that Player O does not need lookahead to win a game with winning condition $L_=$. Next, we give an example in which Player O needs lookahead to win, which is obtained by adapting the exponential lower bound on the necessary lookahead in delay games with safety conditions [7]. In this game, Player O needs exponential lookahead (in the size of an automaton $\mathfrak{A}$ recognizing the winning condition).

Hence, the transition and output function of an explicit transducer have doubly-exponentially-sized domains in $|\mathfrak{A}|$. We show how to construct a succinct transducer of exponential size implementing a winning strategy, an exponential improvement.

Example 6. Consider the reachability automaton $\mathfrak{A}_n$, for $n > 1$, over the alphabet $\Sigma_I \times \Sigma_O = \{1, \dots, n\}^2$ of size $\mathcal{O}(n)$, given in Figure 2. The language of $\mathfrak{A}_n$ contains words of the form $\binom{\alpha}{\beta}$ where $\alpha(1)\alpha(2)\alpha(3)\cdots$ has two occurrences of $\beta(0)$ with only smaller letters in between (a so-called bad j-pair for $j = \beta(0)$). Note that the first letter of α is ignored. In words, the first letter of the second component indicates the existence of a bad j-pair in the α-component (again, without its first letter). It is known that Player O wins $\Gamma^d(L(\mathfrak{A}_n))$ for all $d > 2^n/2$, but not for smaller ones [7].

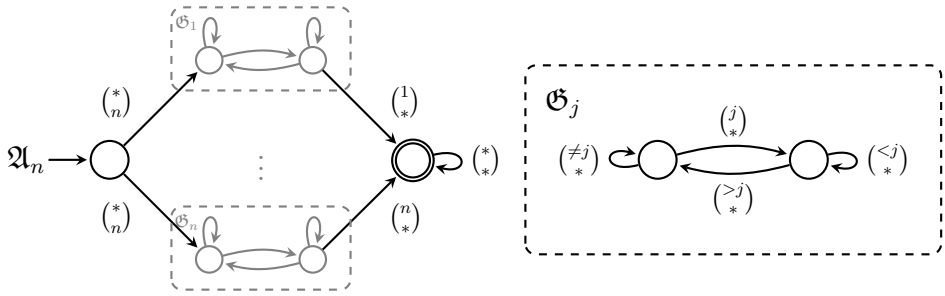

Figure 2: The automaton $\mathfrak{A}_n$ (left) contains gadgets $\mathfrak{G}_1, \dots, \mathfrak{G}_n$ (right). Transitions not depicted lead to a sink state, which is not drawn. The only accepting state is the rightmost state, which is drawn circled. Here, $*$ denotes an arbitrary letter from the respective alphabet.

Now, we show that we can construct a succinct transducer implementing a winning strategy in the block game $\Gamma^d(L(\mathfrak{A}_n))$ of exponential size in n for every $d > 2^n/2$.

To begin with, we note that a block size of $d = 2^n/2 + 1$ is sufficient in order for Player O to win the block game $\Gamma^d(L(\mathfrak{A}_n))$, since every word over $\{1, \dots, n\}$ of length at least 2^n contains a bad j-pair for some j [7].

We now construct a succinct transducer that implements a winning strategy for Player O in the block game $\Gamma^d(L(\mathfrak{A}_n))$ for every $d > 2^n/2$. Clearly, to implement a winning strategy, the output slave of a succinct transducer must identify a bad j-pair before it can make its first output. To achieve this, the following transition structure is used: The automaton collects the

seen letters, upon reading a letter, all smaller seen letters are deleted from the collection, if a letter is seen that has already been collected, a bad pair has been found. More formally, from a state $P \subseteq 2^{\{1,\dots,n\}}$ upon reading the letter j the state $(P \setminus \{1,\dots,j-1\}) \cup \{j\}$ is reached if $j \notin P$, otherwise this is the second occurrence of j in a bad j-pair. Thus, a j-pair can be identified using $\mathcal{O}(2^n)$ states. When a bad j-pair has been found, the output slave produces an output block of length d beginning with j (in a single computation step) and ignores the remaining input.

There are no conditions for subsequent output blocks, in this case the output slave simply copies the input letter by letter until the first \$ occurs and ignores the remaining input.

Thus, the size of an output slave is $\mathcal{O}(2^n)$; the size of a transition slave is constant since it just distinguishes whether the first output block has already been produced. All in all, the constructed succinct transducer is of size $\mathcal{O}(2^n)$.

6.2. Upper Bounds

After these two examples showing how succinct transducers can indeed be smaller than explicit transducers, we prove that they do not have to be larger than explicit ones (when measured in the size of the domains of the transition and output function).

Theorem 3. *Let τ_O be a delay-aware finite-state strategy for a block game $\Gamma^d(L)$ with $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$.*

1. *If τ_O is implementable by an explicit transducer with n states, then also by a succinct transducer with $\mathcal{O}(n \cdot |\Sigma_I|^{2d})$ states.*
2. *If τ_O is implementable by a succinct transducer with n master states, then also by an explicit transducer with n states.*

Proof. 1.) Let $\mathfrak{T} = (Q, \Sigma_I, q_I, \delta, \Sigma_O, \lambda)$ be the explicit transducer implementing τ_O, i.e., $\delta\colon Q \times \Sigma_I^d \to Q$ maps a state and a block in Σ_I^d to a new state and $\lambda\colon Q \times \Sigma_I^d \times \Sigma_I^d \to \Sigma_O^d$ maps a state and two blocks in Σ_I^d to a block in Σ_O^d. The strategy is implemented by a succinct transducer over the same set of master states Q, with the same initial state q_I, and where δ and λ are implemented by slaves Δ and Λ defined below.

The transition slave has states of the form $(q, w) \in Q \times \Sigma_I^{\leq d}$ to store an input block, an initialization function mapping a state $q \in Q$ of $\mathfrak{T}$ to (q, ε), and a transition function mapping a state (q, w) and a letter $a \in \Sigma_I$ to (q, wa), if $|w| < d$. Otherwise, it is mapped to (q, w). This is sufficient, as Δ

is only used to process words of length d. Finally, the output function of the transition slave is defined such that it maps each state (q, w) with $|w| = d$ to $\delta(q, w)$. All other outputs are irrelevant. Then, it is straightforward to prove that the function computed by Δ (restricted to inputs from Σ_I^d) is equal to δ.

The construction of the output slave Λ is analogous: here, we use states of the form $(q, w) \in Q \times \Sigma_I^{\leq 2d}$. Again, the initialization function maps q to (q, ε) and a transition processing a non-\$ input letter appends it to the word stored in the state as long as possible. Furthermore, \$'s are ignored and transitions from states in Σ_I^{2d} can be defined arbitrarily. Transitions processing a \$ from a state of the form $(q, \overline{a_0 a_1})$ output $\lambda(q, \overline{a_0}, \overline{a_1})$, while all other transitions have an empty output. Again, it is straightforward to show that the function computed by Λ coincides with λ on inputs of the form $(q, \overline{a_0}, \overline{a_1})$.

Hence, the succinct transducer constructed using Δ and Λ computes the same function as $\mathfrak{T}$ and has indeed $\mathcal{O}(n \cdot |\Sigma_I|^{2d})$ states.

2.) Now, let $(Q, \Sigma_I, q_I, \Delta, \Sigma_O, \Lambda)$ be a succinct transducer implementing τ_O. Then, τ_O is also implemented by the explicit transducer $(Q, \Sigma_I, q_I, \delta, \Sigma_O, \lambda)$ where $\delta(q, \overline{a})$ is equal to the output of Δ on $\overline{a}$ when initialized with q, and where $\lambda(q, \overline{a_0}, \overline{a_1})$ is the output of Λ on $\overline{a_0}\$\overline{a_1}\$$ when initialized with q. □

Thus, we can obtain a succinct transducer by constructing it starting with an explicit one. This explicit one would typically be obtained by the reduction to Gale-Stewart games presented in Section 4. Next, we show how to turn a finite-state strategy for the Gale-Stewart game into a succinct transducer without the detour via explicit transducers, which yields a smaller transducer.

Theorem 4. *Let $\mathfrak{A}$, $\mathfrak{M}$, $\mathfrak{B}$, and d be as in Theorem 1 or as in Theorem 2.*

If Player O has a finite-state winning strategy for the game $\Gamma(L(\mathfrak{B}))$ with n states, then she has a finite-state winning strategy for $\Gamma^d(L(\mathfrak{A}))$ implemented by a succinct transducer of size $\mathcal{O}(n \cdot |\Sigma_I|^d \cdot d)$.

Proof. Let $Q_{\mathfrak{A}}$ be the state space of $\mathfrak{A}$ and let M be the set of memory states of $\mathfrak{M}$. Furthermore, let $\mathfrak{T} = (Q_{\mathfrak{T}}, R, q_I^{\mathfrak{T}}, \delta_{\mathfrak{T}}, Q_{\mathfrak{A}} \times M_\perp, \lambda_{\mathfrak{T}})$ be a transducer implementing a winning strategy for Player O in $\Gamma(L(\mathfrak{B}))$.

Recall the proof of Item 2 of Theorem 1: there, we turn a finite-state winning strategy for $\Gamma(L(\mathfrak{B}))$ into a delay-aware finite-state winning strategy for $\Gamma^d(L(\mathfrak{A}))$ using a simulation: In $\Gamma(L(\mathfrak{B}))$, Player I picks equivalence

classes from R while Player O picks pairs containing a state of $\mathfrak{A}$ and a memory state of $\mathfrak{M}$. On the other hand, in $\Gamma^d(L(\mathfrak{A}))$, both players pick blocks of letters over their respective alphabet. Now, each block of Player I induces an equivalence class (see Item 1 of Remark 2). For the other direction, we use completions as guaranteed by Remark 1 to translate moves of Player O in $\Gamma(L(\mathfrak{B}))$ into moves of her in $\Gamma^d(L(\mathfrak{A}))$.

Formally, we define a succinct transducer $\mathfrak{T}_s = (Q_{\mathfrak{T}}, \Sigma_I, q_I^{\mathfrak{T}}, \Delta, \Sigma_O, \Lambda)$ simulating $\mathfrak{T}$. To this end, we just need to specify the slaves Δ and Λ.

Intuitively, Δ computes the transition summary of its input, which represents an equivalence class of R, provided the input is long enough. Formally, we define $\Delta = (Q_\Delta, \Sigma_I, q_I^\Delta, \delta_\Delta, Q_{\mathfrak{T}}, \lambda_\Delta)$ where

- $Q_\Delta = Q_{\mathfrak{T}} \times (2^{Q_{\mathfrak{A}} \times M_\perp})^{Q_{\mathfrak{A}}}$,
- $q_I^\Delta(q) = (q, q^* \mapsto \{(q^*, \perp)\})$ for $q \in Q_{\mathfrak{T}}$ and $q^* \in Q_{\mathfrak{A}}$, and
- $\delta_\Delta((q,r),a) = (q,r')$ with $r'(q^*) = \delta_P(r(q^*),a)$ for every $q^* \in Q_{\mathfrak{A}}$, where δ_P is defined as in Section 4 on Page 17.

Thus, we have $\delta_\Delta^*(q_I^\Delta(q), w) = (q, r_w)$. Note that $[\overline{a}]_\equiv$ is an element of R for every block $\overline{a}$ (due to $|\overline{a}| = d$ and Item 1 of Remark 2). Hence, we can define $\lambda_\Delta(q,r) = \delta_{\mathfrak{T}}(q, [w]_\equiv)$ for some w such that $r_w = r$, if such a w exists. If one does exist, then this definition is independent of the choice of w. Otherwise, we define $\lambda_\Delta(q,r)$ arbitrarily. Then, Δ indeed simulates the transition function $\delta_{\mathfrak{T}}$ of $\mathfrak{T}$.

It remains to define the output slave Λ. Note that we need to determine a completion of an input block to simulate the strategy implemented by $\mathfrak{T}$. The *right* completion depends on the block to be completed, not only on its equivalence class. Hence, Λ needs to store the first block in its input using its state space. For the second block, it suffices to determine its equivalence class, which is implemented as in Δ.

Formally, we define $\Lambda = (Q_\Lambda, \Sigma_I, q_I^\Lambda, E_\Lambda, \Sigma_O)$ with

- $Q_\Lambda = Q_{\mathfrak{T}} \times \Sigma_I^{\leq d} \times (2^{Q_{\mathfrak{A}} \times M_\perp})^{Q_{\mathfrak{A}}}$,
- $q_I^\Lambda(q) = (q, \varepsilon, q^* \mapsto \{(q^*, \perp)\})$ for $q \in Q_{\mathfrak{T}}$ and $q^* \in Q_{\mathfrak{A}}$, and
- where E_Λ is defined such that on inputs of the form $w\$w'$ with $|w| = d$ the state $(q, w, r_{w'})$ is reached when initializing the run with q; on all

other inputs an arbitrary state is reached. All these edges have an empty output.

The only non-empty output happens on transitions processing a second \$ from a state of the form (q, w, r) with $|w| = d$ and with $r = r_{w'} \in R$ for some w'. If this is the case, let $(q_0, m_0) = \lambda_{\mathfrak{T}}(\delta_{\mathfrak{T}}(q, [w]_{\equiv}))$ and $(q_1, m_1) = \lambda_{\mathfrak{T}}(\delta_{\mathfrak{T}}(\delta_{\mathfrak{T}}(q, [w]_{\equiv}), [w']_{\equiv}))$. Then, the output of the transition processing \$ from (q, w, r) is some (q_0, q_1, m_1)-completion of w. If such a w' does exist, then this definition is independent of the choice of w'.

Then, Λ indeed simulates the output function $\lambda_{\mathfrak{T}}$ of $\mathfrak{T}$.

Altogether, a straightforward induction as in the proof of Item 2 of Theorem 1 shows that $\mathfrak{T}_s$ indeed implements a winning strategy for the block game. □

6.3. Lower Bounds

After considering upper bounds in the previous two theorems, we now turn our attention to lower bounds showing that the upper bounds are tight for winning conditions recognized by reachability automata. In this case, an exponential lookahead is sufficient and in general necessary [7]. The following construction is an adaption of the lower bound proof for the lookahead, and again based on bad j-pairs.

Example 7. Consider the reachability automaton $\mathfrak{A}_n$, for $n > 1$, over the alphabet $\Sigma_I \times \Sigma_O = (\{1, \ldots, n\} \times \mathbb{B}^n) \times (\{1, \ldots, n\} \times \mathbb{B})$ depicted in Figure 3. The automaton accepts an ω-word

$$\begin{pmatrix} \alpha \\ \beta_1 \\ \vdots \\ \beta_n \\ \gamma \\ \beta \end{pmatrix} \in (\Sigma_I \times \Sigma_O)^\omega$$

with $\alpha, \gamma \in \{1, \ldots, n\}^\omega$ and $\beta_1, \cdots, \beta_n, \beta \in \mathbb{B}^\omega$ if, and only if, it has the following form: there is an m such that $\alpha(1) \cdots \alpha(m)$ contains a bad j-pair for $j = \gamma(0)$, $\alpha(1) \cdots \alpha(m-1)$ contains no bad j-pair (which implies $\alpha(m) = j$), and $\beta_j(0) \cdots \beta_j(m) = \beta(0) \cdots \beta(m)$. Intuitively, Player O has to identify a j such that the α-component of the input contains a bad j-pair

and additionally has to copy the jth β-component up to the end of the first j-pair. Notice that the first letter of the α-component of the input is again ignored when it comes to finding a bad j-pair.

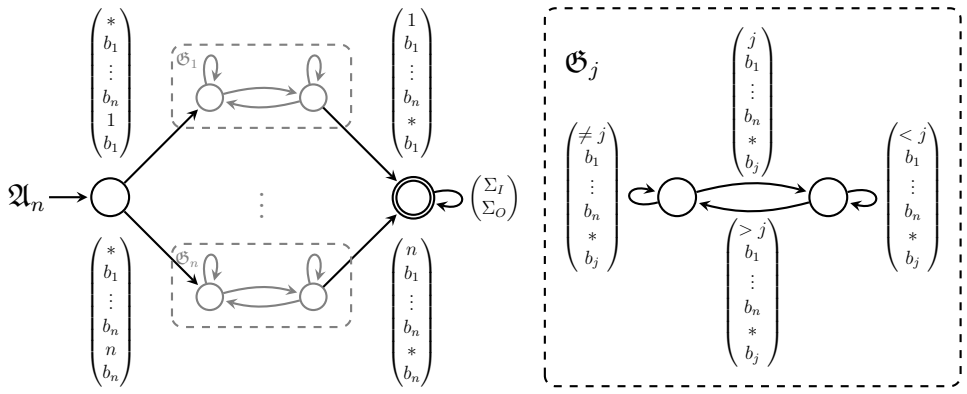

Figure 3: The automaton $\mathfrak{A}_n$ (left) contains gadgets $\mathfrak{G}_1, \ldots, \mathfrak{G}_n$ (right). Transitions not depicted lead to a sink state, which is not drawn. The only accepting state is the rightmost state, which is drawn circled. Here, Σ_I and Σ_O denote an arbitrary letter from the respective alphabet.

Using this example, we can prove the following theorem.

Theorem 5. *For every $n > 1$, there is a language L_n recognized by a reachability automaton $\mathfrak{A}_n$ with $\mathcal{O}(n)$ states such that*

- *Player O has a finite-state winning strategy in the block game $\Gamma^d(L_n)$ for every $d > 2^n/2$, and*
- *every succinct transducer that implements a winning strategy for Player O in the block game $\Gamma^d(L_n)$ for some d has an output slave with at least $\mathcal{O}(2^{n \cdot 2^n})$ states.*

Proof. Consider the reachability automaton $\mathfrak{A}_n$ given in Example 7, let $L_n = L(\mathfrak{A}_n)$. To begin with, we argue that Player O has a finite-state winning strategy in the block game $\Gamma^d(L_n)$ for every $d > 2^n/2$. As already mentioned in Example 6, every word over $\{1, \ldots, n\}$ of length 2^n contains a bad j-pair for some j. A block size of at least $2^n/2 + 1$ allows for a lookahead of at least $2^n + 1$ symbols, thus Player O can correctly identify a bad j-pair by remembering the first two input blocks (recall that the first input letter

is ignored). This observation suffices to implement a finite-state winning strategy adapting the ideas presented in Example 6.

On the other hand, there is a word $x_n \in \{1, \ldots, n\}^*$ of length $2^n - 1$ that has no bad j-pair for every $j \in \{1, \ldots, n\}$ [7]. This allows us to prove that Player O does not win $\Gamma^d(L_n)$ for any $d \leq 2^n/2$: Player I can make the first move in the block game using (a prefix, if necessary, of) the word $1x_n$ in the α-component and any bits in the β-components. Then, Player O has to pick a first letter j^* with her first move (all other choices by her are irrelevant to our argument and thusly ignored). In order to win, she has to pick this j^* so that the input has a bad j^*-pair. However, since by completing x_n and then playing some $j \neq j^*$ ad infinitum, the outcome does not have a bad j^*-pair in its α-component, i.e., Player I wins. For more details, we refer to [7].

We use a generalization of this argument to prove the lower bound on the size of the output slave of a finite-state winning strategy for $\Gamma^d(L_n)$. Hence, let $\mathfrak{T} = (Q, \Sigma_I, q_I, \Delta, \Sigma_O, \Lambda)$ be a succinct transducer that implements a winning strategy in $\Gamma^d(L_n)$. As argued above, we can assume $d > 2^n/2$. Towards a contradiction, assume that the output slave $\Lambda = (Q_\Lambda, \Sigma_I, q_I^\Lambda, E, \Sigma_O)$ has fewer than $2^{n \cdot 2^n}$ states.

Recall that Λ processes words of the form $x_1\$x_2\$$ where $x_1, x_2 \in \Sigma_I^d$ are input blocks. Let X be the set of words of the form

$$\begin{pmatrix} \alpha(0) \cdots \alpha(2^n - 1) \\ \beta_1(0) \cdots \beta_1(2^n - 1) \\ \vdots \\ \beta_n(0) \cdots \beta_n(2^n - 1) \end{pmatrix} \in \Sigma_I^{2^n}$$

with $\alpha(1) \cdots \alpha(2^n - 1) = x_n$. We have $|X| \geq 2^{n \cdot 2^n}$.

Hence, there are two words in X that lead Λ to the same state (when converted into the correct input format for Λ) starting in $q_0 = q_I^\Lambda(q_I)$, which is the initial state used to process the first two blocks. Assume Λ produces an output during these runs. Then, using arguments as above, one can show that it does not implement a winning strategy, as both words do not contain a bad j-pair for any j.

Hence, both runs end in the same state and have not yet produced any output. Thus, if both words are extended by the same suffix, Λ produces the same output for both inputs. Now, let j^* be such that the two words differ in their β_{j^*}-entry at some position. Consider the extension of the two words by picking j^* in the α-component and arbitrary bits in the β-components, until words of length $2d$ are obtained. As both inputs only have bad j-pairs for

$j = j^*$, the automaton has to copy the β_{j^*}-component. However, it cannot achieve this for both inputs, as it is not able to distinguish the different prefixes. Hence, the automaton does not implement a winning strategy. □

A note on the size of the automaton $\mathfrak{A}_n$ for L_n. The number of states is in $\mathcal{O}(n)$, but its alphabet is in $\mathcal{O}(2^n)$. To reduce the size of the alphabet we can consider a variant of L_n defined as follows. We call this variant L'_n, let $\Sigma_I = \Sigma_O = \{1, \dots, n, \mathrm{t}, \mathrm{f}\}$, we use t and f in place of $\mathbb{B}$ to distinguish it from $\{1, \dots, n\}$. We are interested in pairs $\binom{\alpha}{\beta}$ in which the α-component is of the form $\mathrm{a}_0 a_1 w_1 a_2 w_2 \cdots$, where $a_0, a_1, \ldots \in \{1, \dots, n\}$ and $w_1, w_2, \ldots \in \{\mathrm{t}, \mathrm{f}\}^n$. Meaning, instead of vertical n-bit vectors as before, we use horizontal n-bit vectors. If α is not of this form, then every β is allowed in the second component. If α is of this form, then $\binom{\alpha}{\beta} \in L'_n$ if, and only if, β is of the form $b_0 b_1 x_1 b_2 x_2 \cdots$, where $b_0, b_1, b_2 \ldots \in \{1, \dots, n\}$ and $x_0, x_1, \ldots \in \{\mathrm{t}, \mathrm{f}\}^n$ such that if $a_1 \cdots a_i$ is the smallest prefix of $a_1 a_2 \cdots$ that contains a bad j-pair for $j = b_0$, and additionally the first letter of x_k is the jth letter of w_k for $1 \leq k \leq i$.

A reachability automaton $\tilde{\mathfrak{A}}_n$ for L'_n can be constructed with polynomial size in n. The idea is to use an automaton similar to the automaton $\mathfrak{A}_n$, and additionally have a ring counter up to n to compare the first bit of x_k with the jth bit of w_k.

As before, the block game $\Gamma^d(L(\tilde{\mathfrak{A}}_n))$ can be won by Player O for any d that allows enough lookahead to identify a bad j-pair for some j. Since every word over $\{1, \dots, n\}$ of length at least 2^n contains such a pair, every prefix (in the correct format) of length greater than $2^n \cdot (n+1)$ contains such a pair. With the same reasoning as above, a transducer implementing (the output function of) a winning strategy must store every n-bit vector until an occurrence of a bad j-pair for some j has been witnessed. Thus, the state space of such a transducer is in $\mathcal{O}(2^{n \cdot 2^n})$.

6.4. Tradeoff Between Block Size and Memory

Finally, we consider another promising facet of finite-state strategies in delay games: lookahead can be traded for memory and vice versa. Such tradeoffs have previously been presented between lookahead and the semantic quality of winning strategies in games with quantitative winning conditions [1], and between memory size and the semantic quality of winning strategies [37]. With the definition of finite-state strategies, one can add another dimension to the study of tradeoffs in infinite games.

Theorem 6. *For every even $k > 0$, there is a language L_k^R recognized by a safety automaton $\mathfrak{A}_k$ such that*

- *Player O has a finite-state winning strategy in the block game $\Gamma^d(L_k^R)$ for every $d \geq k/2$,*
- *there exists a succinct transducer $\mathfrak{T} = (Q, \Sigma_I, q_I, \Delta, \Sigma_O, \Lambda)$ implementing a winning strategy in $\Gamma^d(L_k^R)$ with $|\Delta| \in \mathcal{O}(2^{k-d})$ and $|\Lambda| \in \mathcal{O}(2^d)$ for every $d \in \{k/2, \ldots, k\}$, and*
- *there exists an explicit transducer $\mathfrak{T} = (Q, \Sigma_I, q_I, \delta, \Sigma_O, \lambda)$ implementing a winning strategy in $\Gamma^d(L_k^R)$ with $|\mathfrak{T}| \in \mathcal{O}(2^{k-d})$ for every $d \in \{k/2, \ldots, k\}$.*

Proof. We start by describing the language L_k^R over the alphabet $\Sigma_I \times \Sigma_O = \mathbb{B}^2$. A pair $\binom{\alpha}{\beta}$ is part of the language if, and only if, $\beta(i) = \alpha((k-1)-i)$ for $0 \leq i \leq k-1$, that is, the first block of length k has to be reversed by Player O.

A safety automaton $\mathfrak{A}_k$ recognizing L_k^R is build as follows. Initially, $\mathfrak{A}_k$ stores the first sequence of length $k/2$ in its state space starting from $(\varepsilon, \uparrow)$ and from a state $\left(\binom{a_1 \cdots a_i}{b_1 \cdots a_i}, \uparrow\right)$ upon reading the next letter $\binom{b_{i+1}}{a_{i+1}}$ it goes to $\left(\binom{a_1 \cdots a_{i+1}}{b_1 \cdots b_{i+1}}, \uparrow\right)$ for $0 \leq i \leq k/2 - 1$. Say $\mathfrak{A}_k$ has reached $\left(\binom{a_1 \cdots a_{k/2}}{b_1 \cdots b_{k/2}}, \uparrow\right)$, then upon reading the letter $\binom{a_{k/2}}{b_{k/2}}$ it goes to the state $\left(\binom{a_1 \cdots a_{k/2-1}}{b_1 \cdots b_{k/2-1}}, \downarrow\right)$; and to a rejecting sink with any other letter. Subsequently, it has to check whether the next sequence of length $k/2 - 1$ is equal to $\binom{b_{k/2-1} \cdots b_1}{a_{k/2-1} \cdots a_1}$. This can be done checking that in a state $\left(\binom{a_1 \cdots a_i}{b_1 \cdots b_i}, \downarrow\right)$ the next read letter is $\binom{b_i}{a_i}$ and going to $\left(\binom{a_1 \cdots a_{i-1}}{b_1 \cdots b_{i-1}}, \downarrow\right)$ for $1 \leq i \leq k/2 - 1$. After reaching $(\varepsilon, \downarrow)$, any sequence is valid.

It is easy to see that Player O can win the block game for every $d \geq k/2$. Now, for $d \in \{k/2, \ldots, k\}$, we show that there exists a succinct transducer $\mathfrak{T} = (Q, \Sigma_I, q_I, \Delta, \Sigma_O, \Lambda)$ implementing a winning strategy for Player O in $\Gamma^d(L_k^R)$ with $|\Delta| \in \mathcal{O}(2^{k-d})$ and $|\Lambda| \in \mathcal{O}(2^d)$. Let $x\gamma \in \Sigma_I^\omega$ with $x = a_1 \cdots a_k \in \Sigma_I^k$ denote the input sequence that Player I plays in the block game $\Gamma^d(L_k^R)$. The first output block that must be produced by Player O is $a_k \cdots a_{k-d-1}$. This sequence is part of the first lookahead, the output slave Λ of a succinct transducer $\mathfrak{T}$ has to store this sequence completely to reverse

it, thus $|\Lambda| \in \mathcal{O}(2^d)$. The next output block that has to be produced must begin with $a_{k-d} \cdots a_1$. This sequence is not part of the next lookahead, it is part of the first input block (the first $k-d$ letters to be precise), the next lookahead is the second and third input block. Thus, is must be stored by the transition slave Δ of $\mathfrak{T}$ so that this sequence can be passed on to Λ which has to output it. Hence, Δ has to memorize the first $k-d$ input letters, resulting in $|\Delta| \in \mathcal{O}(2^{k-d})$.

Regarding explicit transducers, the same reasoning can be applied. Thus, in order to implement a winning strategy in the block game $\Gamma^d(L_k^R)$, an explicit transducer $\mathfrak{T} = (Q, \Sigma_I, q_I, \delta, \Sigma_O, \lambda)$ has to memorize the first $k-d$ input letters, resulting in a state space of size $\mathcal{O}(2^{k-d})$. Recall, $|Q|$ is defined as the size of $\mathfrak{T}$, hence $|\mathfrak{T}| \in \mathcal{O}(2^{k-d})$.

Taking a look at the special cases of $d = k/2$ and $d = k$, the above result yields that an explicit transducer $\mathfrak{T}$ needs memory of $\mathcal{O}(2^{k/2})$ and in the latter case no memory to win the block game $\Gamma^d(L_k^R)$. Generally, for some d between $k/2$ and k, an explicit transducer $\mathfrak{T}$ needs memory of $\mathcal{O}(2^{k-d})$ to implement a winning strategy in the block game $\Gamma^d(L_k^R)$. Let us analyze this result; increasing the block size by one halves the number of memory states an explicit transducer needs, thus the tradeoff between the block size and the necessary memory is gradual. □

The example of the block-reversal winning condition L_k^R presented in the proof of Theorem 6 allows for a tradeoff between the block size and the necessary memory to implement a winning strategy in the block game. However, the size of an automaton that recognizes L_k^R as well as the lower bound on the block size is exponential in k, so the necessary lookahead is only linear in the size of the automaton. It is an open question whether there is a winning condition recognizable by an automaton of polynomial size with an exponential lower bound on the necessary block size that allows for a tradeoff between block size and memory.

7. Conclusion

We have presented a very general framework for analyzing delay games. If the automaton recognizing the winning condition satisfies a certain aggregation property, our framework yields upper bounds on the necessary lookahead to win the game, an algorithm for determining the winner (under some additional assumptions on the acceptance condition), and finite-state

winning strategies for Player O, if she wins the game at all. These results cover all previous results on the first two aspects (although not necessarily with optimal complexity of determining the winner).

Thereby, we have lifted another important aspect of the theory of infinite games to the setting with delay. However, many challenging open questions remain, e.g., a systematic study of memory requirements in delay games is now possible. For delay-free games, tight upper and lower bounds on these requirements are known for almost all winning conditions.

Furthermore, in our study we focussed on the state complexity of the automata implementing the strategies, i.e., we measure the quality of a strategy in the number of states of a transducer implementing it. However, this is not the true size of such a machine, as this ignores the need to represent the transition function and the output function, which have an exponential domain (in the block size) in the case of delay-aware strategies. We addressed this issue and have proposed a succinct notion of transducers implementing delay-aware strategies. Although we have presented examples where our succinct notion allows for a significantly smaller representation of strategies compared to the true size of an explicit representation, generally such a representation cannot be smaller than an explicit one.

Another exciting question concerns the tradeoff between memory and amount of lookahead: can one trade memory for lookahead? In other settings, such tradeoffs exist, e.g., lookahead allows to improve the quality of strategies [1]. We have presented a game where Player O can indeed trade lookahead for memory and vice versa. Salzmann has presented further tradeoffs of this kind, e.g., linear lookahead allows exponential reductions in memory size in comparison to delay-free strategies [14]. In current work, we investigate whether these results are inherent to his notion of finite-state strategy, which differs subtly from the one proposed here, or whether they exist in our setting as well.

Finite-state strategies in arena-based games are typically computed by game reductions, which turn a game with a complex winning condition into one in a larger arena with a simpler winning condition. In future work, we plan to lift this approach to delay games. Note that the algorithm for computing finite-state strategies presented here can already be understood as a reduction, as we turn a delay game into a Gale-Stewart game. This removes the delay, but preserves the type of winning condition. However, it is also conceivable that staying in the realm of delay games yields better results, i.e., by keeping the delay while simplifying the winning condition. In

future work, we address this question.

Acknowledgements. The authors are very grateful to the anonymous reviewers of this and an earlier version of the paper, which significantly improved the exposition.

[1] M. Zimmermann, Games with costs and delays, in: LICS 2017 [38], pp. 1–12. doi:10.1109/LICS.2017.8005125.

[2] M. Chen, M. Fränzle, Y. Li, P. N. Mosaad, N. Zhan, What's to come is still unsure - synthesizing controllers resilient to delayed interaction, in: S. K. Lahiri, C. Wang (Eds.), ATVA 2018, Vol. 11138 of LNCS, Springer, 2018, pp. 56–74. doi:10.1007/978-3-030-01090-4_4.

[3] D. Gale, F. M. Stewart, Infinite games with perfect information, Annals of Mathematics 28 (1953) 245–266. doi:10.1515/9781400881970-014.

[4] F. A. Hosch, L. H. Landweber, Finite delay solutions for sequential conditions, in: ICALP 1972, 1972, pp. 45–60.

[5] J. R. Büchi, L. H. Landweber, Solving sequential conditions by finite-state strategies, Transactions of the American Mathematical Society 138 (1969) 295–311. doi:10.2307/1994916.

[6] M. Holtmann, L. Kaiser, W. Thomas, Degrees of lookahead in regular infinite games, LMCS 8 (3). doi:10.2168/LMCS-8(3:24)2012.

[7] F. Klein, M. Zimmermann, How much lookahead is needed to win infinite games?, LMCS 12 (3). doi:10.2168/LMCS-12(3:4)2016.

[8] D. A. Martin, Borel determinacy, Annals of Mathematics 102 (1975) 363–371. doi:10.2307/1971035.

[9] F. Klein, M. Zimmermann, What are strategies in delay games? Borel determinacy for games with lookahead, in: S. Kreutzer (Ed.), CSL 2015, Vol. 41 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 519–533. doi:10.4230/LIPIcs.CSL.2015.519.

[10] W. Fridman, C. Löding, M. Zimmermann, Degrees of lookahead in context-free infinite games, in: M. Bezem (Ed.), CSL 2011, Vol. 12 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011, pp. 264–276. doi:10.4230/LIPIcs.CSL.2011.264.

[11] F. Klein, M. Zimmermann, Prompt delay, in: A. Lal, S. Akshay, S. Saurabh, S. Sen (Eds.), FSTTCS 2016, Vol. 65 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 43:1–43:14. doi:10.4230/LIPIcs.FSTTCS.2016.43.

[12] M. Zimmermann, Delay games with WMSO+U winning conditions, RAIRO - Theor. Inf. and Applic. 50 (2) (2016) 145–165. doi:10.1051/ita/2016018.

[13] I. Walukiewicz, Pushdown processes: Games and model-checking, Inf. and Comput. 164 (2) (2001) 234–263. doi:10.1006/inco.2000.2894.

[14] T. Salzmann, How much memory is needed to win regular delay games?, Master's thesis, Saarland University (2015).

[15] M. Zimmermann, Finite-state strategies in delay games, in: P. Bouyer, A. Orlandini, P. S. Pietro (Eds.), GandALF 2017, Vol. 256 of EPTCS, 2017, pp. 151–165. doi:10.4204/EPTCS.256.11.

[16] M. O. Rabin, Decidability of second-order theories and automata on infinite trees, Transactions of the American Mathematical Society 141 (1-35) (1969) 4.

[17] B. Khoussainov, Finite state strategies in one player McNaughton games, in: C. Calude, M. J. Dinneen, V. Vajnovszki (Eds.), DMTCS 2003, Dijon, Vol. 2731 of LNCS, Springer, 2003, pp. 203–214. doi:10.1007/3-540-45066-1_16.

[18] A. Rabinovich, Synthesis of finite-state and definable winning strategies, in: R. Kannan, K. N. Kumar (Eds.), FSTTCS 2009, Vol. 4 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009, pp. 359–370. doi:10.4230/LIPIcs.FSTTCS.2009.2332.

[19] S. L. Roux, A. Pauly, Extending finite memory determinacy to multiplayer games, in: A. Lomuscio, M. Y. Vardi (Eds.), SR 2016, Vol. 218 of EPTCS, 2016, pp. 27–40. doi:10.4204/EPTCS.218.3.

[20] W. Thomas, Finite-state strategies in regular infinite games, in: P. S. Thiagarajan (Ed.), FSTTCS 1994, Vol. 880 of LNCS, Springer, 1994, pp. 149–158. doi:10.1007/3-540-58715-2_121.

[21] S. Dziembowski, M. Jurdziński, I. Walukiewicz, How much memory is needed to win infinite games?, in: LICS 1997, IEEE Computer Society, 1997, pp. 99–110. doi:10.1109/LICS.1997.614939.

[22] K. Chatterjee, T. A. Henzinger, F. Horn, The complexity of request-response games, in: A. H. Dediu, S. Inenaga, C. Martín-Vide (Eds.), LATA 2011, Vol. 6638 of LNCS, Springer, 2011, pp. 227–237. doi:10.1007/978-3-642-21254-3_17.

[23] F. Horn, Streett games on finite graphs, in: GDV 2005, 2005. URL `https://www.irif.fr/ horn/publications.html`

[24] N. Wallmeier, P. Hütten, W. Thomas, Symbolic synthesis of finite-state controllers for request-response specifications, in: O. H. Ibarra, Z. Dang (Eds.), CIAA 2003, Vol. 2759 of LNCS, Springer, 2003, pp. 11–22. doi:10.1007/3-540-45089-0_3.

[25] N. Fijalkow, F. Horn, Les jeux d'accessibilité généralisée, Technique et Science Informatiques 32 (9-10) (2013) 931–949, see also [39]. doi:10.3166/tsi.32.931-949.

[26] T. Colcombet, N. Fijalkow, F. Horn, Playing safe, in: V. Raman, S. P. Suresh (Eds.), FSTTCS 2014, Vol. 29 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014, pp. 379–390. doi:10.4230/LIPIcs.FSTTCS.2014.379.

[27] E. A. Emerson, C. S. Jutla, Tree automata, mu-calculus and determinacy (extended abstract), in: FOCS 1991, IEEE Computer Society, 1991, pp. 368–377. doi:10.1109/SFCS.1991.185392.

[28] A. Mostowski, Games with forbidden positions, Tech. Rep. 78, University of Gdańsk (1991).

[29] C. S. Calude, S. Jain, B. Khoussainov, W. Li, F. Stephan, Deciding parity games in quasipolynomial time, in: H. Hatami, P. McKenzie, V. King (Eds.), STOC 2017, ACM, 2017, pp. 252–263. doi:10.1145/3055399.3055409.

[30] J. Fearnley, S. Jain, S. Schewe, F. Stephan, D. Wojtczak, An ordered approach to solving parity games in quasi polynomial time and quasi

linear space, in: H. Erdogmus, K. Havelund (Eds.), SPIN 2017, ACM, 2017, pp. 112–121. doi:10.1145/3092282.3092286.

[31] M. Jurdzinski, R. Lazic, Succinct progress measures for solving parity games, in: LICS 2017 [38], pp. 1–9. doi:10.1109/LICS.2017.8005092.

[32] K. Lehtinen, A modal μ perspective on solving parity games in quasi-polynomial time, in: A. Dawar, E. Grädel (Eds.), LICS 2018, ACM, 2018, pp. 639–648. doi:10.1145/3209108.3209115.

[33] M. Bojańczyk, Weak MSO with the unbounding quantifier, Theory Comput. Syst. 48 (3) (2011) 554–576. doi:10.1007/s00224-010-9279-2.

[34] R. McNaughton, Infinite games played on finite graphs, Ann. Pure Appl. Logic 65 (2) (1993) 149–184. doi:10.1016/0168-0072(93)90036-D.

[35] A. Pnueli, R. Rosner, On the synthesis of an asynchronous reactive module, in: G. Ausiello, M. Dezani-Ciancaglini, S. R. D. Rocca (Eds.), ICALP 1989, Vol. 372 of LNCS, Springer, 1989, pp. 652–671. doi:10.1007/BFb0035790.

[36] E. Grädel, W. Thomas, T. Wilke (Eds.), Automata, Logics, and Infinite Games: A Guide to Current Research, Vol. 2500 of LNCS, Springer, 2002. doi:10.1007/3-540-36387-4.

[37] A. Weinert, M. Zimmermann, Easy to win, hard to master: Optimal strategies in parity games with costs, LMCS 13 (3). doi:10.23638/LMCS-13(3:29)2017.

[38] 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017, IEEE Computer Society, 2017.

[39] N. Fijalkow, F. Horn, The surprizing complexity of reachability games, arXiv 1010.2420.
URL `http://arxiv.org/abs/1010.2420`

Appendix A. Arena-based Games vs. Gale-Stewart Games

In this short appendix, we give a formal definition of the arena-based games mentioned in Section 3. We begin by giving a quick recap of arena-based games to introduce our notation.

An arena $\mathcal{A} = (V, V_I, V_O, E, v_I)$ consists of a finite directed graph (V, E) without terminal vertices, a partition (V_I, V_O) into the positions of Player I and Player O, and an initial vertex $v_I \in V$. A play is an infinite path through $\mathcal{A}$ starting in v_I.

A game $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ consists of an arena $\mathcal{A}$, say with set V of vertices, and a winning condition $\mathrm{Win} \subseteq V^\omega$. A play is winning for Player O, if it is in Win.

A strategy for Player O is a mapping $\sigma\colon V^*{\cdot}V_O \to V$ such that $(v, \sigma(wv)) \in E$ for every $wv \in V^*V_O$. A play $v_0v_1v_2\cdots$ is consistent with σ, if $v_{i+1} = \sigma(v_0 \cdots v_i)$ for every i with $v_i \in V_O$. A strategy is winning, of every consistent play is winning for Player O. If Player O has a winning strategy for $\mathcal{G}$, then we say she wins $\mathcal{G}$.

A finite-state strategy for an arena $\mathcal{A}$ with set V of vertices is again implemented by a transducer $\mathfrak{T} = (Q, \Sigma_I, q_I, \delta, \Sigma_O, \lambda)$ where Q, q_I, and δ are as in Subsection 2.3, where $\Sigma_I = \Sigma_O = V$, and $\lambda\colon Q \times V \to V$. The strategy implemented by $\mathfrak{T}$ is defined as $\sigma(wv) = \lambda(\delta^*(q_I, wv), v)$, where $\delta^*(q_I, wv)$ is the state reached by $\mathfrak{T}$ when processing wv starting in q_I. The size of $\mathfrak{T}$ is defined to be $|Q|$.

A strategy is finite-state if it is implemented by some finite transducer; it is positional, if it is implemented by some transducer of size one.

Now, given a Gale-Stewart game $\Gamma(L(\mathfrak{A}))$ for some automaton $\mathfrak{A} = (Q, \Sigma_I \times \Sigma_O, q_I, \delta, \mathrm{Acc})$, we define the arena-based game $\mathcal{G}_\mathfrak{A} = (\mathcal{A}_\mathfrak{A}, \mathrm{Win}_\mathfrak{A})$ with $\mathcal{A}_\mathfrak{A} = (V, V_I, V_O, E, v_I)$ such that:

- $V = V_I \cup V_O$ with $V_I = \delta \cup \{v_I\}$ for some fresh initial vertex $v_I \notin \delta$ and $V_O = Q \times \Sigma_I$.
- E is the union of the following sets of edges:
  - $\{(v_I, (q_I, a)) \mid a \in \Sigma_I\}$ (initial moves of Player I),
  - $\{((q, \binom{a}{b}, q'), (q', a')) \mid (q, \binom{a}{b}, q') \in V_1, a' \in \Sigma_I\}$ (regular moves of Player I), and
  - $\{((q, a), (q, \binom{a}{b}, q')) \mid (q, a) \in V_0, b \in \Sigma_O, q' = \delta(q, \binom{a}{b})\}$ (moves of Player O).

- $\mathrm{Win}_{\mathfrak{A}} = \{v_I(q_0, a_0)t_0(q_1, a_1)t_1(q_2, a_2)t_2 \cdots \mid t_0t_1t_2 \cdots \in \mathrm{Acc}\}$.

The following lemma formalizes a claim from Section 3.

Lemma 4. *Let $\Gamma(L(\mathfrak{A}))$ and $\mathcal{G}_{\mathfrak{A}}$ be defined as above. Then, Player O wins $\Gamma(L(\mathfrak{A}))$ if, and only if, she wins $\mathcal{G}_{\mathfrak{A}}$. Furthermore, a finite-state winning strategy with n states for Player O in $\mathcal{G}_{\mathfrak{A}}$ can be turned into a finite-state winning strategy with $|Q| \cdot |\Sigma_I| \cdot n$ states for Player O in $\Gamma(L(\mathfrak{A}))$.*

Proof. There is a bijection between play prefixes in $\Gamma(L(\mathfrak{A}))$ and in $\mathcal{G}_{\mathfrak{A}}$. By taking limits, this bijection can be lifted to a bijection between plays that additionally preserves the winner of plays. Using the former bijection one can easily translate strategies between these games and use the second bijection to prove that this transformation preserves being a winning strategy. Finally, it is also straightforward to implement the transformation from $\mathcal{G}_{\mathfrak{A}}$ to $\Gamma(L(\mathfrak{A}))$ with finite-state strategies: the transducer implementing the strategy for $\Gamma(L(\mathfrak{A}))$ uses a product state space consisting of the states of the given transducer for $\mathcal{G}_{\mathfrak{A}}$ and Player O vertices from $\mathcal{G}_{\mathfrak{A}}$ to keep track of the last vertex of the play prefix obtained by the first bijection. This information is sufficient to mimic the strategy for $\mathcal{G}_{\mathfrak{A}}$ in $\Gamma(L(\mathfrak{A}))$. □

Now, for some delay game $\Gamma_f(L(\mathfrak{A}))$ for some automaton $\mathfrak{A} = (Q, \Sigma_I \times \Sigma_O, q_I, \delta, \mathrm{Acc})$ with constant delay function f with $f(0) = d > 0$, we define the arena-based game $\mathcal{G}_{\mathfrak{A},d} = (\mathcal{A}_{\mathfrak{A},d}, \mathrm{Win}_{\mathfrak{A},d})$ with $\mathcal{A}_{\mathfrak{A},d} = (V, V_I, V_O, E, v_I)$ such that:

- $V = V_I \cup V_O$ with $V_I = \delta \times \Sigma_I^{d-1} \cup \{v_I\}$ for some fresh initial vertex v_I and $V_O = Q \times \Sigma_I^d$.
- E is the union of the following sets of edges:
  - $\{(v_I, (q_I, w)) \mid w \in \Sigma_I^d\}$ (initial moves of Player I),
  - $\{(((q, \binom{a}{b}, q'), w), (q', wa')) \mid ((q, \binom{a}{b}, q'), w) \in V_1, a' \in \Sigma_I\}$ (regular moves of Player I), and
  - $\{((q, aw), (q, \binom{a}{b}, q'), w) \mid (q, aw) \in V_0, b \in \Sigma_O, q' = \delta(q, \binom{a}{b})\}$ (moves of Player O).
- $\mathrm{Win}_{\mathfrak{A}} = \{v_I(q_0, w_0)(t_0, w_0')(q_1, w_1)(t_1, w_1')(q_2, w_2)(t_2, w_2') \cdots \mid t_0t_1t_2 \cdots \in \mathrm{Acc}\}$.

Again, the following lemma formalizes a claim from Section 3.

Lemma 5. *Let $\Gamma_f(L(\mathfrak{A}))$ and $\mathcal{G}_{\mathfrak{A},d}$ be defined as above. Then, Player O wins $\Gamma_f(L(\mathfrak{A}))$ if, and only if, she wins $\mathcal{G}_{\mathfrak{A},d}$. Furthermore, a finite-state winning strategy with n states for Player O in $\mathcal{G}_{\mathfrak{A},d}$ can be turned into a finite-state winning strategy with $|Q| \cdot |\Sigma_I|^d \cdot n$ states for Player O in $\Gamma_f(L(\mathfrak{A}))$.*

Proof. Similarly to the proof of Lemma 4. □