The Complexity of Request-Response Games

Krishnendu Chatterjee[1], Thomas A. Henzinger[1], and Florian Horn[1,2]

[1] IST (Institute of Science and Technology), Austria
{krish.chat,tah}@ist.ac.at
[2] LIAFA, CNRS & Université Paris 7, France
horn@liafa.jussieu.fr

Abstract. We consider two-player graph games whose objectives are request-response condition, *i.e* conjunctions of conditions of the form "if a state with property Rq is visited, then later a state with property Rp is visited". The winner of such games can be decided in EXPTIME and the problem is known to be NP-hard. In this paper, we close this gap by showing that this problem is, in fact, EXPTIME-complete. We show that the problem becomes PSPACE-complete if we only consider games played on DAGs, and NP-complete or PTIME-complete if there is only one player (depending on whether he wants to enforce or spoil the request-response condition).

We also present near-optimal bounds on the memory needed to design winning strategies for each player, in each case.

1 Introduction

Games. Games played on graphs are suitable models for multi-component systems: vertices represent states; edges represent transitions; players represent components; and objectives represent specifications. The specification of a component is typically given as an ω-regular condition [6], and the resulting ω-regular games have been used for solving control and verification problems (see, e.g., [1,7,8]).

Fairness specifications. The class of fairness objectives is one of the most important specifications in verification and synthesis. The two classical notions of fairness are as follows: (a) strong fairness (or Streett) objectives, and (b) request-response (or assume-guarantee) objectives. The fairness objectives consist of a set of k pairs of requests and corresponding responses. The Streett objective requires that every request that appears infinitely often must be granted infinitely often. The request-response objective requires that every request that appears is granted after it appears. The class of Streett objectives is a canonical and widely used form of fairness specification [10,6]. The class of request-response (assume-guarantee) specifications was studied in [11], and it was shown that a wide range of practical specifications (such as an elevator controller) can be specified as request-response specifications.

Previous results. Games with Streett objectives have been widely studied and optimal bounds on computational complexity and memory required by winning

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 227–237, 2011.

strategies have been established. The winner problem in games with Streett objectives with k request-response pairs is coNP-complete [5]. The memory bound for winning strategies is as follows: there is an optimal (matching lower and upper) bound of $k!$ for the size of memory for the player with the Streett objective and the opposing player has a memoryless winning strategy (a strategy that is independent of the history and depends only on the current vertex) [4]. In contrast, for games with request-response objectives there are gaps both in the computational complexity bounds, and memory bounds required for winning strategies. Games with request-response objectives can be solved in EXPTIME [11] and are NP-hard [3]. The winning strategies for the player with request-response objective require a memory of size at least $\lfloor k/3 \rfloor \cdot 2^{\lfloor k/3 \rfloor}$ and memory of size $k \cdot 2^{k+1}$ suffices for winning strategies for both players [11].

Our results. We present tight computational complexity bounds for request-response games, and present near optimal bounds on memory required by winning strategies. Our results are as follows:

1. We first show that games with request-response objectives are EXPTIME-complete (improving the NP-hardness lower bound). In the study of turn-based deterministic games with classical objectives such as Rabin, Streett, Muller the complexities are NP-complete, coNP-complete, PSPACE-complete, respectively [10]. For turn-based games, several EXPTIME-completeness results are known for more general class of games such as pushdown games [12] or imperfect-information games [9]. We show that for perfect-information finite-state turn-based deterministic games, a natural variant of Streett objectives lead to EXPTIME-completeness. The EXPTIME-hardness results for pushdown or imperfect-information games are either due to the infinite store (stack) or the imperfect-information, whereas our proof is different and shows how to exploit the simple extension of Streett objectives to request-response objectives to mimic runs of alternating polynomial space Turing machines.
2. For the special class of DAG-games we show that request-response objectives are PSPACE-complete. We also study the complexity of one player game graphs: if there is only one player with request-response objectives, then the problem is NP-complete; and if there is only the opposing player, then the problem can be solved in polynomial time.
3. We improve the lower bounds for memory required for winning strategies in games with request-response objectives: we show that the protagonist player (whose goal is to enforce the request-response objective) requires $2^k - 1$ and the antagonist (opposing) player requires 2^k memory states, (improving the lower bound of $\lfloor k/3 \rfloor \cdot 2^{\lfloor k/3 \rfloor}$ for the protagonist player, and no bound was known for the opposing player). With a very simple argument we show the construction of [11] can be used to obtain an upper bound $k \cdot 2^k$ for the protagonist and 2^k for the antagonist. Thus our lower bound of $2^k - 1$ almost matches the upper bound of $k \cdot 2^k$ for the protagonist, and our bound of 2^k for the opposing player is a tight bound. Thus, we present almost optimal bounds on memory required by winning strategies. For DAG-games

with request-response objectives, we prove an optimal (matching upper and lower) bound of memory size $\binom{k}{\lfloor k/2 \rfloor}$ for winning strategies of both players.

2 Definitions

Arenas and plays. A (finite) game *arena* $\mathcal{A}$ is a tuple $((\mathcal{V}, \mathcal{E}), \mathcal{V}_\circ, \mathcal{V}_\Box)$, where $(\mathcal{V}, \mathcal{E})$ is a finite graph and $\mathcal{V}_\circ, \mathcal{V}_\Box$ is a partition of $\mathcal{V}$. The vertices in $\mathcal{V}_\circ$ are called *Eve's vertices* and those in $\mathcal{V}_\Box$ are called *Adam's vertices*. For a vertex u in $\mathcal{V}$, we denote by $\mathcal{E}(u)$ the set of successors of u: $\mathcal{E}(u) = \{v \in \mathcal{V} \mid (u, v) \in \mathcal{E}\}$. We assume that every vertex has at least one successor. A play ρ on an arena $\mathcal{A}$ is a (possibly infinite) sequence $\rho = \rho_0, \rho_1, \ldots$ of vertices respecting the edge relation: for all $i \geq 0$ we have $(\rho_i, \rho_{i+1}) \in \mathcal{E}$.

Strategies. Intuitively, a strategy is a recipe to extend plays. Formally, a *strategy* σ for Eve is a function $\sigma : \mathcal{V}^* \cdot \mathcal{V}_\circ \to \mathcal{V}$ such that for all finite plays (or histories) x ending in a vertex v of Eve, $\sigma(x)$ is a successor of v. Strategies for Adam are defined analogously (and are usually denoted τ).

An equivalent definition of strategies uses the notion of *memory*. A *strategy with memory* σ for Eve is a tuple $(\sigma^{\mathtt{M}}, \sigma^{\mathtt{i}}, \sigma^{\mathtt{n}}, \sigma^{\mathtt{u}})$ where $\sigma^{\mathtt{M}}$ is the set of *memory states*, $\sigma^{\mathtt{i}} \in \sigma^{\mathtt{M}}$ is the *initial memory state*, $\sigma^{\mathtt{n}} : \mathcal{V}_\circ \times \sigma^{\mathtt{M}} \to \mathcal{V}$ is the *next-move function*, and $\sigma^{\mathtt{u}} : \mathcal{V} \times \sigma^{\mathtt{M}} \to \sigma^{\mathtt{M}}$ is the *memory update function*. Notice that any strategy can be represented as a strategy with memory $\mathcal{V}^*$. A strategy σ has finite memory if $\sigma^{\mathtt{M}}$ is finite (in this case, $|\sigma^{\mathtt{M}}|$ is the *size of* σ); it is memoryless if $\sigma^{\mathtt{M}}$ is a singleton. Notice that a memoryless strategy for Eve is independent of the history of the play and depends only on the current vertex, and hence can be described as a function from $\mathcal{V}_\circ$ to $\mathcal{V}$ respecting the edge relation. The notation for strategies with memory and memoryless strategies for Adam is analogous.

A play ρ is *consistent with* σ if for all $i \geq 0$ such that ρ_i belongs to Eve we have ρ_{i+1} is $\sigma(\rho_0, \rho_1, \ldots, \rho_i)$. Given an initial vertex $v \in \mathcal{V}$, a strategy σ for Eve and a strategy τ for Adam, we denote by $\rho(v, \sigma, \tau)$ the unique infinite play starting in v and consistent with σ and τ.

Request-response objectives. A *winning condition (objective)* Φ for an arena $\mathcal{A}$ is a subset of the plays on the arena. In this paper, we consider the request-response objectives introduced by Wallmeier, Hütten, and Thomas in [11]. It refers to vertex properties $\mathsf{Rq}_1, \ldots, \mathsf{Rq}_k$ which capture k different "requests", and other vertex properties $\mathsf{Rp}_1, \ldots, \mathsf{Rp}_k$ which represent the corresponding "responses" (each $\mathsf{Rq}_i, \mathsf{Rp}_i \subseteq \mathcal{V}$). The associated request-response condition requires that *for each i, whenever a vertex in Rq_i is visited, then later a vertex in Rp_i is visited.* In linear time temporal logic (LTL) the condition is more often formalized as $\wedge_{i=1}^{k} \mathrm{G}(\mathsf{Rq}_i \to \mathrm{XF}(\mathsf{Rp}_i))$, where G, X, and F denote *globally*, *next*, and *eventually*, respectively. The Streett objective in LTL is described as $\wedge_{i=1}^{k} (\mathrm{G\ F}(\mathsf{Rq}_i) \to \mathrm{G\ F}(\mathsf{Rp}_i))$.

A strategy σ is *winning for Eve from a vertex v* in a game $\mathcal{G} = (\mathcal{A}, \Phi)$ if, for any strategy τ for Adam, the play $\rho(v, \sigma, \tau)$ belongs to Φ. A strategy τ is winning for Adam from a vertex v if for all strategies σ, the play $\rho(v, \sigma, \tau)$ belongs to $\neg\Phi = \Pi \setminus \Phi$. The *winning region of Eve in* $\mathcal{G}$, denoted $W_E(\Phi)$, is the

set of vertices from which Eve has a winning strategy, and the winning region for Adam, denoted $W_A(\neg\Phi)$, is defined similarly.

Theorem 1 (Determinacy [11]). *For all request-response games, for all vertices v, either Eve or Adam has a winning strategy from v.*

3 Complexity of Request-Response Games

In this section, we consider the computational complexity of request-response games in general. Our main result is an EXPTIME lower bound in complexity, matching the EXPTIME membership from [11]. We also study the complexity of the winning strategies in terms of memory, and provide near optimal bounds for both players.

3.1 Request-Response Games Are EXPTIME-Complete

In [11], the authors show that request-response games can be solved in EXPTIME, but they do not provide any lower bound in complexity. In this subsection, we show that the problem is in fact EXPTIME-hard, through a reduction from the membership problem for alternating polynomial space Turing machines.

An *alternating Turing machine* (ATM) is a tuple $(Q, q_{\mathsf{in}}, Q_\vee, Q_\wedge, \mathcal{I}, \delta, q_{\mathsf{acc}})$ where:

- Q is a finite set of control states, partitioned into existential ($Q_\vee$) and universal ($Q_\wedge$) states;
- $q_{\mathsf{in}} \in Q$ is the initial state;
- $\mathcal{I} = \{0, 1\}$ is the tape alphabet;
- $\delta \subseteq Q \times \mathcal{I} \times Q \times \mathcal{I} \times \{-1, 1\}$ is the transition relation;
- $q_{\mathsf{acc}} \in Q$ is the accepting state.

For a given polynomial p, the question of whether an ATM M accepts a word w in space at most $p(|w|)$ is EXPTIME-complete [2]. We reduce this problem to the winner problem of request-response games. The idea is that the players build a run of the machine: Eve controls the existential states and Adam the universal ones; if the run reaches an accepting state, Eve wins; if it goes on forever, Adam does. A winning strategy for Eve in the game translates as an accepting run tree of the machine.

We use $p(|w|)$ copies of the control graph of the machine in order to store the current location of the head. However, the arena does not store the *content* of the tape. Instead, at each step, Eve announces the current symbol and Adam either accepts it or challenges it. If he does the latter, the play stops: Eve wins if she has been truthful; Adam wins if she cheated. This interaction is described in Figure 1.

We use request-response pairs to force Eve to announce the correct symbol at each step. There is a pair ℓ^s for each location ℓ and each symbol s. An extra pair \$ guarantees that the correct simulation of an infinite run is winning for Adam.

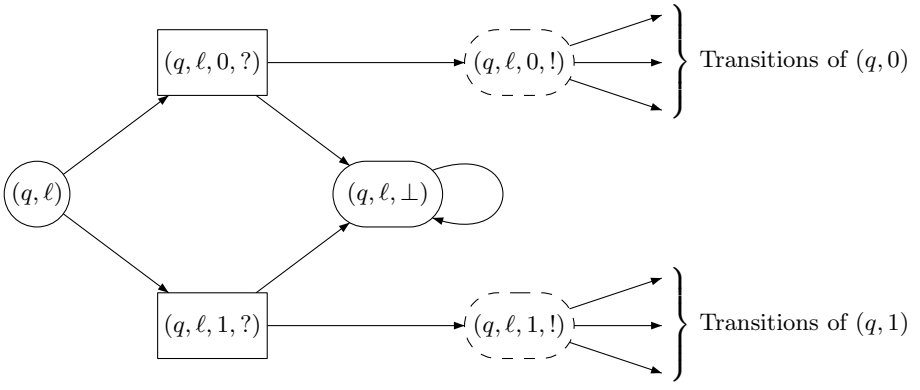


Fig. 1. The consistency gadget at a state (q, ℓ)

The idea is that whenever a symbol s is written on the location ℓ, a request of type ℓ^s is generated. The play begins in the vertex start, which is a request of type i^{w_i} for each $i \in \{0, \ldots, |w| - 1\}$, as well as a request of type \$. Then the token goes to $(q_{\mathsf{in}}, 0)$ for the first step.

At the beginning of a step where the machine is in the control state q and its head is at the ℓ^{th} cell, the token is in the vertex (q, ℓ), which belongs to Eve. Her first task is to announce the contents of the cell. She does so by granting either ℓ^0 or ℓ^1 (by going to $(q, \ell, 0, ?)$ or $(q, \ell, 1, ?)$, respectively). At this point, Adam can challenge her choice by sending the token to $(q, \ell, \bot)$, which is a sink where all the pairs except for ℓ^0 and ℓ^1 are granted. Thus, if Eve has announced the correct symbol, she wins; otherwise, either ℓ^0 or ℓ^1 is left pending and she loses. If Adam chooses to accept Eve's claim, the token goes to the vertex $(q, \ell, i, !)$, which belongs to Eve if q is an existential state and to Adam if q is a universal state. There, the controlling player chooses a transition of the form $t = (q, i, r, j, \pm)$ by going to the vertex (t, ℓ), which generates a request of type ℓ^j. The token goes then to the vertex $(r, \ell \pm 1)$ for the next step, unless $r = q_{\mathsf{acc}}$, in which case the token goes to the sink vertex stop, which grants all the requests.

Let us show that Eve has a winning strategy in $\mathcal{G}$ if, and only if, M accepts w. We call *honest* a strategy for Eve which always calls the correct symbol in vertices of the form (q, ℓ), and *trusting* a strategy for Adam which never challenges the choices of Eve. It is clear that any winning strategy of Eve has to be honest, and that an honest strategy of Eve is winning if and only if it is winning against any trusting strategy of Adam.

There is a natural bijection between (i) plays consistent with an honest strategy for Eve and a trusting one for Adam and (ii) runs of M on w. It can be extended to a bijection between the honest strategies of Eve and the run trees of M on w, which matches winning strategies and accepting run trees. Thus Eve has a winning strategy if and only if M accepts w. It follows that the problem of deciding the winner in request-response games is `EXPTIME`-hard. As it is known to belong to `EXPTIME` [11], Theorem 2 follows:

Theorem 2. *The problem of deciding the winner in a request-response game is* EXPTIME*-complete.*

3.2 Strategy Complexity

We consider now the complexity, in terms of memory, of the winning strategies for both players. In [11], the reduction to Büchi games yields strategies with memory $k \cdot 2^k$. It is possible to improve these bounds a little with two simple observations:

- In the original reduction, the arena keeps track of the pending requests (2^k memory) and of an "active" pair which must be satisfied next (k memory); Eve does not need to discriminate between the vertices where the active pair is not pending, so she only needs $\sum_{i=0}^{k} i \cdot \binom{k}{i} = k \cdot 2^{k-1}$.
- By replacing the Büchi condition with a generalized Büchi conditions (with k target sets), one can get rid of the "active pair" tracker; Adam still has memoryless winning strategies in the reduced game, so he only needs 2^k memory states in the original request-response game.

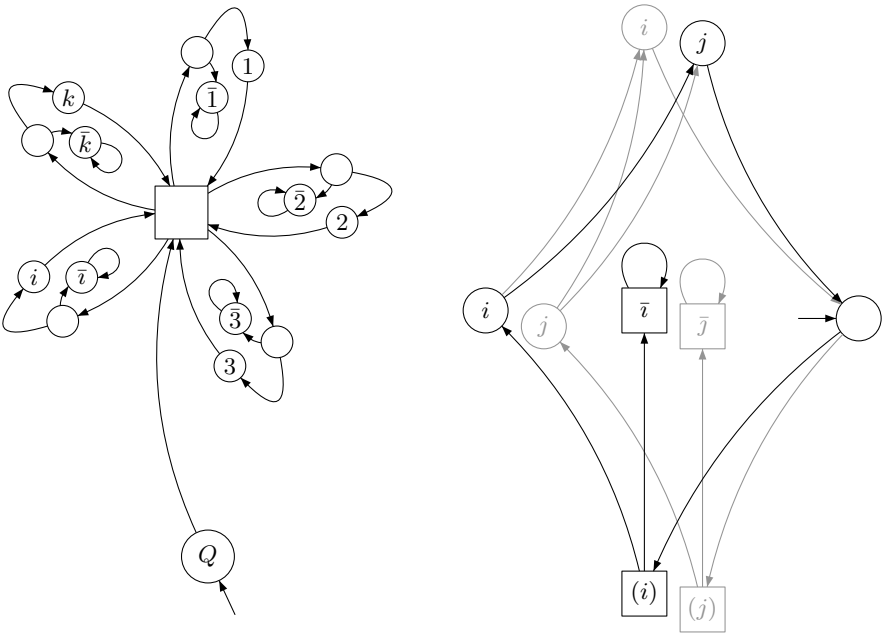
The authors presented only a lower bound for Eve, who was shown to need $2^{\lfloor k/3 \rfloor}$ memory states. In this section, we improve and complete this picture with a better lower bound for Eve ($2^k - 1$), and a tight bound for Adam (2^k). The games realizing the lower bounds are presented in Figure 2.

In the game of Figure 2(a), the vertex labelled Q is a request of each type; for each i, a vertex labelled i is a response of type i, and a vertex labelled $\bar{i}$ is a response of every type but i. Intuitively, a play is divided in steps in which Adam first chooses a pair and Eve then grants either this pair (and the play continues) or all the others (and the play stops). It is clear that Eve can win with the following strategy: the first time Adam chooses the i^{th} petal, she grants the pair i; the second time, she grants all the other pairs[1]. We show that Adam can defeat any strategy with less than $2^k - 1$ memory states.

Let $\sigma = (\sigma^{\mathtt{M}}, \sigma^{\mathtt{i}}, \sigma^{\mathtt{n}}, \sigma^{\mathtt{u}})$ be a strategy for Eve with less than $2^k - 1$ memory states. For each memory state m, we define the stopping set $\chi(m)$ of m as the set of petals where Eve would stop the play if Adam chose them (notice that m is the memory of Eve in the "heart" vertex: it might change after Adam has made his choice, but her behaviour is still determined by m and the petal that Adam chose). As there are less than $2^k - 1$ memory states, there is a strict subset X of $\{1, \ldots, k\}$ which is not the stopping set of any memory state. Now, Adam can win against σ by choosing, at each step, a petal in the symmetric difference of X and $\chi(m)$, where m is Eve's current memory under σ. Such a play can either go on forever if Adam keeps to petals in X, or stop the first time he gets out. In either case, there is at least one request outside of X which is never granted.

In the game of Figure 2(b), the arena has $4k + 1$ vertices: there is one copy of the bottom, middle, left, and top vertices for each request-response pair.

[1] This strategy uses 2^k memory states, but it is clear that there is no actual need for a specific memory state to remember that every petal has been visited: in this case, the play is already won, no matter what Eve does later on.



(a) Eve: $2^k - 1$ (b) Adam: 2^k

Fig. 2. Lower bounds in memory

For each pair, say the i^{th}, the vertices labelled i) (left and top) are both requests and responses (recall that requests have to be granted in the strict future); the vertex labelled $\bar{\imath}$ (middle) is a response of every type but i. The right and bottom vertices are neither requests nor responses of any type (the label (i) of the bottom vertex only serves as a reminder that there are k different copies of this vertex). From the initial vertex (on the right), Eve can go to any of the bottom vertices; likewise, from *each* left vertex, Eve can go to any of the top vertices. By contrast, in a bottom vertex, say (i), Adam has to go to either the left vertex i or the middle vertex $\bar{\imath}$. A step of this game can be described by the three following actions: Eve chooses a pair, say i; Adam either grants it and requests it again (and the play continues), or grants every other pair (and the game stops); Eve then chooses a (possibly different) pair, say j, grants it, and requests it again. Adam can win by stopping the game the second time a pair is requested (thus with 2^k memory states), and we show that he cannot win with less.

Let $\tau = (\tau^{\mathtt{M}}, \tau^{\mathtt{i}}, \tau^{\mathtt{n}}, \tau^{\mathtt{u}})$ be a strategy for Adam with less than 2^k memory states. For a memory state m in $\tau^{\mathtt{M}}$, we define the stopping set $\chi(m)$ of m as the sets of bottom vertices where Adam would stop the play if Eve chose them (once again, m is the memory of Adam in the right vertex: it might change after Eve's choice, but Adam's behaviour is determined by m and this choice). As there are less than 2^k memory states, there is a subset X of $\{1, \dots, k\}$ which is not the stopping set of any memory state. Now, Eve can win against τ by choosing, at

each step, a pair in the symmetric difference of X and $\chi(m)$, where m is Adam's current memory under τ in the right vertex and cycling through the pairs in X in the left one. Such a play can either go on forever if Eve keeps to petals in X, or stop the first time she gets out. In both cases, only requests in X are enabled. Furthermore, in the first case, each such request is granted infinitely often; in the second case, each request in X is granted when the play stops, and never enabled again.

Theorem 3. *In any request-response game with k request-response pairs, wherever Eve has a winning strategy, she has a winning strategy with memory $k \cdot 2^{k-1}$; wherever Adam has a winning strategy, he has a winning strategy with memory 2^k. Furthermore, there is a request-response game with k request-response pairs in which Eve can only win with at least $2^k - 1$ memory states, and one where Adam can only win with at least 2^k memory states.*

4 Restrictions

In this section, we consider two special types of request-response games, where the winner problem is easier to solve.

4.1 DAG Arenas

The first one is the case where the arenas have the form of a directed acyclic graph (no cycles apart from loops on vertices with no other successors). By contrast to the usual study of "long-term" behaviours, these games focus on "short-term" objectives. We show that request-response games played on DAG-arenas are PSPACE-complete, and provide tight bounds for the memory required of each player.

As with most games played on DAG arenas, it is possible to solve the winner problem in polynomial space, by enumerating the plays in lexicographic order. We show PSPACE-hardness through a reduction from the truth problem of quantified boolean formulae. From a QBF in conjunctive normal form with k variables, we derive a request-response game with $3 \cdot k + 1$ vertices as follows: there is a vertex for each variable *and* one for each literal; the "variable" vertex leads to the two corresponding "literal vertices, and belongs to Eve if the variable is existential or to Adam if is is universal; there is a request-response pair for each clause, which is requested at the beginning of the play and solved at each literal present in the clause. For a QBF of the form $\exists x_1, \forall x_2, \ldots, \exists x_k$, the resulting game is described in Figure 3 (the vertex $\mathcal{C}$ is a request of each type).

Theorem 4 follows:

Theorem 4. *The problem of deciding the winner in request-response games played on DAG arenas is* PSPACE*-complete.*

The restriction to DAG arenas also affects the complexity of strategies. Theorem 5 provides tight bounds for both players:

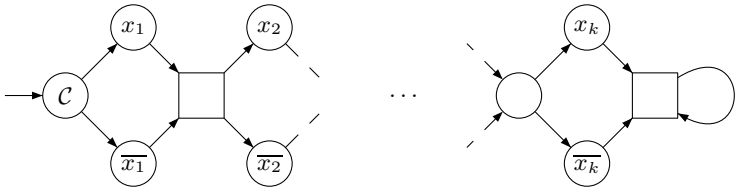


Fig. 3. Reduction from QBF to request-response games on DAG arenas

Theorem 5. *In any request-response game with k request-response pairs played on a DAG-arena, wherever a player has a winning strategy, he has a winning strategy with memory $\binom{k}{\lfloor k/2 \rfloor}$. Furthermore, for each player there is a request-response game with k request-response pairs in they can only win with at least $\binom{k}{\lfloor k/2 \rfloor}$ memory states.*

Proof. In order to devise a winning strategy for either player, it is enough to remember the current set of unanswered requests: as all the plays are finite, they are winning if, and only if, they end with all their requests answered. Furthermore, there is no need to keep two separate memory states for two sets A and B of pairs such that $A \subset B$: if Eve can win in both cases, she can do so in both cases by playing as if the set of unanswered requests was B; symmetrically, Adam can win by playing in both cases as if the set of unanswered requests was A. As there are at most $\binom{k}{\lfloor k/2 \rfloor}$ incomparable subsets of $\{1, \ldots, k\}$, both Eve and Adam can win with memory $\binom{k}{\lfloor k/2 \rfloor}$ in any request-response game with k request-response pairs.

A family of arenas where this much memory is necessary can be described as follows:

- *Eve.* Adam can choose $\lfloor k/2 \rfloor$ requests. Then Eve can choose $\lfloor k/2 \rfloor$ responses. It is clear that she must choose the exact the same subset that Adam chose. As there are $\binom{k}{\lfloor k/2 \rfloor}$ possibilities, she needs memory $\binom{k}{\lfloor k/2 \rfloor}$.
- *Adam.* All pairs are initially requested. Eve can choose $\lfloor k/2 \rfloor$ responses, then Adam chooses $\lfloor k/2 \rfloor$ requests. Finally, Eve can choose $k-1$ responses. Again, Adam needs to match the subset that Eve chose, so he needs $\binom{k}{\lfloor k/2 \rfloor}$ memory states.

Theorem 5 follows. □

4.2 One-Player Arenas

One-player games correspond to the synthesis of controllable systems, with no interaction from the environment. Game problems are usually much simpler in this case. For example, if the player tries to ensure a request-response specification, the winner problem becomes NP-complete:

Theorem 6. *The problem of deciding whether Eve has a winning strategy in a one-player request-response game is* NP*-complete.*

Proof. We can reduce SAT to one-player games using the same reduction that we used in the former section: if all the quantifiers are existential, all the vertices in the resulting game belong to Eve. Thus the problem is NP-hard.

In order to describe a NP procedure to solve this problem, first observe that a play always consists of a finite path w followed by infinite occurrences of all the vertices in a strongly connected component C. It is winning for Eve if every request unresolved in w or present in C is matched by a corresponding response in C. The crux of the proof is the fact that we can always choose w of size at most $(k+1) \cdot |\mathcal{V}|$ by removing from it all the cycles which do not contain the *last* occurrence of a response. We can thus guess non-deterministically both w and C, and the NP-membership follows. □

If the player is trying to spoil, rather than ensure, a request-response objective, the winner problem can be decided in polynomial time:

Theorem 7. *The problem of deciding whether Adam has a winning strategy in a one-player request-response game is* PTIME*-complete.*

Proof. The PTIME hardness comes from the trivial reduction from alternating reachability. In order to describe a PTIME procedure, observe that in order to win, Adam needs only to reach a request from which he can avoid the corresponding response. As safety and reachability winning regions can be computed in polynomial time, so can be the winning region of Adam in a one-player game where he controls all the vertices. □

References

1. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. JACM 49, 672–713 (2002)
2. Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
3. Chatterjee, K., Henzinger, T., Horn, F.: Finitary winning in ω-regular games. ACM ToCL 11(1) (2009)
4. Dziembowski, S., Jurdziński, M., Walukiewicz, I.: How much memory is needed to win infinite games? In: Logic In Computer Science, pp. 99–110. IEEE Computer Society, Los Alamitos (1997)
5. Emerson, E., Jutla, C.: The complexity of tree automata and logics of programs. In: Foundations of Computer Science, pp. 328–337. IEEE Computer Society, Los Alamitos (1988)
6. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Heidelberg (1992)
7. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL, pp. 179–190. ACM, New York (1989)
8. Ramadge, P., Wonham, W.: Supervisory control of a class of discrete-event processes. SIAM Journal of Control and Optimization 25, 206–230 (1987)

9. Reif, J.H.: The complexity of two-player games of incomplete information. Journal of Computer and System Sciences 29(2), 274–301 (1984)
10. Thomas, W.: Languages, automata, and logic. Handbook of Formal Languages 3, 389–455 (1997)
11. Wallmeier, N., Hütten, P., Thomas, W.: Symbolic synthesis of finite-state controllers for request-response specifications. In: Ibarra, O.H., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 11–22. Springer, Heidelberg (2003)
12. Walukiewicz, I.: Pushdown processes: Games and model-checking. Inf. Comput. 164(2), 234–263 (2001)