Memory Reduction via Delayed Simulation

Marcus Gelderie Michael Holtmann
{gelderie,holtmann}@automata.rwth-aachen.de
RWTH Aachen University, Lehrstuhl für Informatik 7, D-52056 Aachen

We address a central (and classical) issue in the theory of infinite games: the reduction of the memory size that is needed to implement winning strategies in regular infinite games (i.e., controllers that ensure correct behavior against actions of the environment, when the specification is a regular ω-language). We propose an approach which attacks this problem before the construction of a strategy, by first reducing the game graph that is obtained from the specification. For the cases of specifications represented by "request-response"-requirements and general "fairness" conditions, we show that an exponential gain in the size of memory is possible.

1 Introduction

Infinite games are a tool for the construction and verification of reactive systems. We consider the case of two players, *Player 0* modeling a controller and *Player 1* its environment. We deal with finite arenas and regular winning conditions, the latter one being captured by standard automata theoretic acceptance conditions (for example Büchi or Muller conditions). For these types of games the winner is computable and a finite-state winning strategy can be constructed [1, 6, 13, 5].

There are many criteria for measuring the quality of a winning strategy. If only a finite memory is needed, then we mostly consider the size of this memory. This view has been pursued in many papers, among them [2]. For example, it is known that weak and strong Muller games over a graph with n vertices can be solved with winning strategies of size at most $\mathcal{O}(2^n)$ and $\mathcal{O}(n!)$, respectively. There are well-known examples that show the optimality of these bounds [2].

A standard method for the construction of winning strategies is to proceed in two steps. In a first step, the game graph G is expanded by a memory structure S, yielding a larger game graph G' (loosely indicated as $G' = S \times G$), while the original winning condition φ is transformed into a simpler one (φ'), allowing for positional winning strategies over G'. In the case of weak Muller games the memory structure is the powerset of the set of all vertices of G, whereas in strong Muller games we consider the set of all sequences of vertices of G. From a positional winning strategy over G' one immediately obtains a finite-state winning strategy over G with memory S. A reduction of the memory size can then be performed by classical minimization algorithms for sequential functions (as they are computed by Mealy automata).

In this paper we pursue an alternative approach that addresses the aspect of memory reduction at an earlier stage, namely *before* the construction of a positional winning strategy over G'. More precisely, we insert an intermediate step of reducing G', viewing it as an acceptor $\mathcal{A}$ of an ω-language, where the winning condition φ' is used as acceptance condition. This reduction yields a smaller graph G'_0 with memory structure S_0; it has the same type of winning condition as G'. For the graph G'_0 we construct a positional winning strategy, which is subsequently transformed into a finite-state winning strategy with memory structure S_0 over G.

The main challenge in this approach is to introduce a method for reducing ω-automata with acceptance conditions that are known to define games with positional winning strategies. State space reduction

Johannes Reich and Bernd Finkbeiner (Eds): International Workshop on Interactions, Games and Protocols (iWIGP)
EPTCS 50, 2011, pp. 46–60, doi:10.4204/EPTCS.50.4

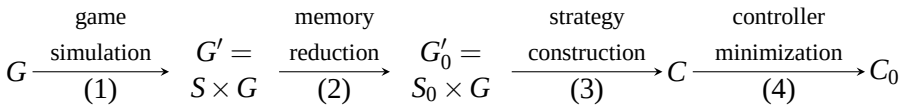

Figure 1: Our Memory Reduction Approach (see Step 2)

of ω-automata is a difficult problem, already for Büchi conditions. In [7] our method was presented for the case of weak Muller games. Reduction of weak parity automata can be done by the method of minimization presented in [8]. It turned out that the approach can result in an exponential gain, regarding the size of the memory needed.

In the present paper we address the more difficult case of "strong" winning conditions. Whereas a weak winning condition merely refers to visits and non-visits to vertices, a strong winning condition considers the set of vertices visited infinitely often. We deal with two particular winning conditions of practical interest, namely Request-Response and Streett conditions.

In a Request-Response game the winning condition is a conjunction of statements "whenever a state with property p is visited, then sometime later a state with property q". Formally, we are given a set $\Omega = \{(P_1,R_1),\dots,(P_k,R_k)\}$ of pairs of subsets of V. Player 0 wins if every request, i.e., a visit to some P_j, is eventually followed by a matching response, i.e., the set R_j is visited sometime later. A Streett winning condition is denoted similarly, we call the sets E_j, F_j. Player 0 wins if infinitely many visits to F_j imply infinitely many visits to E_j.

We propose a method to reduce the size of ω-automata with the aforementioned types of acceptance conditions, using this to reduce the size of game graphs before the construction of winning strategies. For the case of Büchi automata we apply the approach of "delayed" simulation presented in [3]. A state q is delayed simulated by a state q' if, in a run from q, each visit to a final state can eventually be answered by a visit to a final state in a run from q'. This condition is tested via a simulation game between two players. For the parity condition, as obtained by a game simulation of a Streett game, we use an extended version of delayed simulation [4]. If in the run from the simulated state a particular color is seen, then in the run from the simulating state this color has to be exceeded by a better one (regarding the acceptance condition). In our setting, computation of delayed simulation for the Büchi condition can be reduced to minimization of standard DFA, whereas for the parity condition we have to solve the corresponding simulation game explicitly.

This work is structured as follows: In the subsequent section we recall the basic terminology and known results. Section 3 presents our approach in abstract terms. The main issue here is to reconcile the two views of a graph as used for the presentation of an infinite game and for the definition of an ω-language. Section 4 develops the approach for the case of request-response games and shows an example where an exponential gain in the size of the memory needed for implementing a winning strategy is obtained. In Section 5 we treat analogously the case of Streett games.

2 Preliminaries

An *infinite game* $\Gamma = (G,\varphi)$ is played by two players, Player 0 and Player 1. The *game arena* is a finite directed graph $G = (V,E)$ with each vertex belonging to either player, i.e., $V = V_0 \cup V_1$ where V_0 belongs to Player 0 and V_1 belongs to Player 1, and $E \subseteq V \times V$. The *winning condition* $\varphi \subseteq V^\omega$ is the set of all infinite paths through G which are winning for Player 0. Starting from an initial vertex the players move

a token along edges in E, building up an infinite *play* $\rho = \rho(0)\rho(1)\rho(2)\cdots$. If the current vertex belongs to V_0, then Player 0 moves the token, and analogously for Player 1. The play ρ is *winning for Player 0* if $\rho \in \varphi$, otherwise it is winning for Player 1.

A *strategy* for Player 0 is a function f defining a next move for every game position of Player 0 (analogously for Player 1), i.e., it is a partial function $f : V^*V_0 \to V$ such that for every play prefix $v_0 \cdots v_k$ with $v_k \in V_0$ it holds $(v_k, f(v_0 \cdots v_k)) \in E$. A play $\rho = \rho(0)\rho(1)\rho(2)\cdots$ is *played according to* f if for all $\rho(i) \in V_0$ it holds $\rho(i+1) = f(\rho(0)\cdots\rho(i))$. A strategy f is called a *winning strategy from v* for Player 0 if each play starting in v that is played according to f is winning for Player 0. The *winning region* W_0 of Player 0 is the set of all vertices from where Player 0 has a winning strategy. Strategies can be implemented by I/O-automata in the format of Mealy-automata.

In this work we deal with *Request-Response* and *Streett* games. A request-response winning condition is given by a set $\Omega = \{(P_1,R_1),\ldots,(P_k,R_k)\}$ of pairs of subsets of V. A request is a visit to a set P_j, and a response is a visit to a set R_j, for $1 \leq j \leq k$. A play $\rho = \rho(0)\rho(1)\rho(2)\cdots \in V^\omega$ is winning for Player 0 if and only if every request is eventually responded to, i.e., for every j it holds

$$\forall i(\rho(i) \in P_j \Longrightarrow \exists i' \geq i : \rho(i') \in R_j)$$

A Streett winning condition is induced by a set $\Omega = \{(E_1,F_1),\ldots,(E_k,F_k)\}$ of pairs of subsets of V. For a play ρ, let $\mathrm{Inf}(\rho)$ be the set of vertices visited infinitely often in ρ. The play is winning for Player 0 if and only if for every pair (E_j,F_j) it holds

$$\mathrm{Inf}(\rho) \cap F_j \neq \varnothing \Longrightarrow \mathrm{Inf}(\rho) \cap E_j \neq \varnothing.$$

For the algorithm we are going to introduce we need the notion of *game simulation*. The key idea of a game simulation is to extend the given game graph by a memory component such that on the new game graph a simpler winning condition can be used to simulate the original one. Any solution to the extended game can be used to compute an I/O-automaton that implements a winning strategy for the original game (see for example [10]).

Definition 1. Let $\Gamma = (G,\varphi)$ and $\Gamma' = (G',\varphi')$ be infinite games with game graphs $G = (V,E)$ and $G' = (V',E')$ and winning conditions φ, φ'. We say that Γ is *simulated* by Γ' (short: $\Gamma \leq \Gamma'$) if and only if the following hold:

1. $V' = S \times V$ for a finite memory set S (and $(s,v) \in V'_i \iff v \in V_i$)
2. There exists $s_0 \in S$ such that every play ρ of Γ is transformed into a unique play ρ' of Γ' by
   (a) $\rho(0) = v \Longrightarrow \rho'(0) = (s_0,v)$
   (b) Let $(s,v) \in V'$:
      i. $(v,v') \in E \Longrightarrow \exists s' \in S : ((s,v),(s',v')) \in E'$
      ii. $((s,v),(s_1,v_1)) \in E', ((s,v),(s_2,v_2)) \in E' \Longrightarrow s_1 = s_2$
   (c) $((s,v),(s',v')) \in E' \Longrightarrow (v,v') \in E$
3. ρ is winning for Player 0 in $\Gamma \iff \rho'$ is winning for Player 0 in Γ'

Later on, we present game simulation algorithms for request-response games by *Büchi* games and for Streett games by *parity* games [12, 5]. A Büchi winning condition is induced by a set $F \subseteq V$ of final vertices. A play ρ is winning for Player 0 if and only if the set F is visited infinitely often: $F \cap \mathrm{Inf}(\rho) \neq \varnothing$. We solve a Büchi game as follows: First, we compute the set $\mathrm{Recur}_0(F) \subseteq F$ consisting of all final vertices from where Player 0 can force infinitely many visits to F. Afterwards, we compute the 0-Attractor of $\mathrm{Recur}_0(F)$, i.e., the set of all vertices from where Player 0 can force a visit to $\mathrm{Recur}_0(F)$.

It can be shown that this attractor coincides with the winning region of Player 0 in the Büchi game and that an associated attractor winning strategy reduces the distance to F in each move (cf. Π^0_2-games in [10]). A parity winning condition is given by a *coloring* of the set of vertices, i.e., a function $c : V \to \{0,\dots,m\}$. A play ρ is winning for Player 0 if and only if the maximal color seen infinitely often, denoted $\max(c(\mathrm{Inf}(\rho)))$, is even.

All types of games we consider are *determined*: from each vertex one of the players has a winning strategy. In the sequel, we denote a Büchi game (G,F) rather than (G,φ), and analogously for other types of games. We assume that the reader is familiar with the basic theory of ω-automata (see for example [11, 5]).

3 Reduction of Game Graphs

The idea of our memory reduction algorithm is to reduce the game graph G' before computing a winning strategy. To get this in a formal setting we transform infinite games into ω-automata, and vice versa. We view the simulating game Γ' as an ω-automaton $\mathscr{A}$ accepting exactly the plays winning for Player 0 in Γ. The automaton $\mathscr{B}$ is obtained from $\mathscr{A}$ by state space reduction in such a way that the structural properties of game simulation are preserved, i.e., Γ is simulated by Γ'', where Γ'' is the automaton $\mathscr{B}$ viewed as infinite game. To reduce $\mathscr{A}$ we compute a language-preserving equivalence relation on the memory S.

Definition 2. Let $\Gamma = (G,\varphi)$ and $\Gamma' = (G',\varphi')$ be infinite games such that $\Gamma \leq \Gamma'$. We define the (deterministic) *game automaton* $\mathscr{A} = ((S \times V) \cup \{q_0, q_{\mathrm{sink}}\}, q_0, \delta, \psi, V_0)$ over V. The function δ is adopted from E' and a transition is labeled by the V-component of its target state. For $v' \in V$ we set $\delta(q_0,v') := (s_0,v')$ and $\delta(q_{\mathrm{sink}},v') := q_{\mathrm{sink}}$. For $s \in S, v,v' \in V$ with $(v,v') \notin E$ we set $\delta((s,v),v') := q_{\mathrm{sink}}$. The acceptance condition ψ is defined on an abstract level: A run $q_0\rho'$ of $\mathscr{A}$ is defined accepting if and only if ρ' is a winning play for Player 0 in Γ'. (Conversely, an *automaton game* is constructed from a game automaton in the obvious way. For that we need to keep V_0 in $\mathscr{A}$.)

A simulating game and its game automaton are equivalent in the following sense.

Remark 1. Let Γ,Γ' be infinite games such that $\Gamma \leq \Gamma'$ and $\mathscr{A}$ the game automaton of Γ'. Then, $\mathscr{A}$ accepts exactly the plays winning for Player 0 in Γ: $L(\mathscr{A}) = \varphi$.

We now reduce the game automaton in such a way that the properties of game simulation are preserved. To retain item 1 from Definition 1 we compute an equivalence relation $\approx$ on $S \times V$ and refine it to $\approx_S$ on S, where only $\approx_S$ is used for reduction. Moreover, to achieve item 3 we have to preserve the language of $\mathscr{A}$. We require the following structural properties for $\approx$.

Definition 3. Let $\mathscr{A}$ be a game automaton and let $\approx$ be an equivalence relation on $S \times V$. We say that $\approx$ is *compatible* with $\mathscr{A}$ if and only if the following hold:

1. For all $s_1,s_2 \in S, v,v' \in V$:
   $(s_1,v) \approx (s_2,v) \Longrightarrow \delta((s_1,v),v') \approx \delta((s_2,v),v')$
2. Let ρ and ρ' be two runs in $\mathscr{A}$ (starting at arbitrary states) such that it holds $\rho(i) \approx \rho'(i)$, for all $i \in \mathbb{N}$. Then ρ is accepting if and only if ρ' is accepting.

The quotient automaton of $\mathscr{A}$ with respect to $\approx_S$ is defined on the basis of the following observation: If $(s_1,v) \approx (s_2,v)$ holds then from these two states exactly the same inputs are accepted. (Note that $\mathscr{A}$ gets as inputs the plays of the game Γ.) If this is true for all $v \in V$, then s_1 and s_2 can be considered equivalent.

Definition 4. Let $\mathscr{A}$ be a game automaton and let $\approx$ be a compatible equivalence relation on $S \times V$. The equivalence relation $\approx_S$ on S is defined as follows:

$$s_1 \approx_S s_2 : \Longleftrightarrow \forall v \in V : (s_1, v) \approx (s_2, v)$$

For $s \in S$, $[s]$ denotes the equivalence class of s with respect to $\approx_S$. Given $s_1 \approx_S s_2$ and $(v, v') \in E$, let $(s_i', v') := \delta((s_i, v), v')$ for $i = 1, 2$. According to Definition 3 we know that $(s_1', v') \approx (s_2', v')$ holds. However, $s_1' \approx_S s_2'$ does not hold necessarily. To get the v'-successor of $([s_1], v)$ well-defined we use some fixed total order $\prec_S$ on S.

Definition 5. Let $\approx$ be compatible and $\approx_S$ be derived from it as above. We define the *quotient automaton* $\mathscr{A}/_{\approx_S} = ((S/_{\approx_S} \times V) \cup \{q_0, q_{\text{sink}}\}, q_0, \delta/_{\approx_S}, \psi/_{\approx_S}, V_0)$ over V. Given $([s], v) \in S/_{\approx_S} \times V$ and $(v, v') \in E$ we define

$$\delta/_{\approx_S}(([s], v), v') := ([s_{\min}], v')$$

where

$$s_{\min} := \min\{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ and } \delta((\hat{s}, v), v') = (\hat{s}', v')\}.$$

The rest of $\delta/_{\approx_S}$ is defined analogously. Let $\rho = q_0([s_1], v_1)([s_2], v_2) \cdots$ be a run of $\mathscr{A}/_{\approx_S}$. We define ρ to be accepting if and only if the run $\rho' = q_0(s_1', v_1)(s_2', v_2) \cdots$ of $\mathscr{A}$ (which is uniquely determined by ρ) is accepting.

The run ρ' is uniquely determined by ρ because both $\mathscr{A}$ and $\mathscr{A}/_{\approx_S}$ are deterministic. The acceptance condition for $\mathscr{A}/_{\approx_S}$ immediately implies $L(\mathscr{A}) = L(\mathscr{A}/_{\approx_S})$. Later on, we show that for reducing Büchi game automata and parity game automata there exist compatible equivalence relations which are computable efficiently from a game automaton $\mathscr{A}$. Moreover, the respective quotient automaton $\mathscr{A}/_{\approx_S}$ can be defined with the same type of acceptance condition, in both cases. The following theorem shows that the automaton game Γ'' of $\mathscr{A}/_{\approx_S}$ has the same structural properties as Γ', i.e., Γ is simulated by Γ''. (For the proof see [7].)

Theorem 1 ([7])**.** *Let $\Gamma = (G, \varphi)$ and $\Gamma' = (G', \varphi')$ be infinite games such that $\Gamma \leq \Gamma'$. Let $\mathscr{A}$ be the game automaton of Γ' and $\approx$ a compatible equivalence relation on $S \times V$. Then Γ is simulated by the automaton game Γ'' of $\mathscr{A}/_{\approx_S}$.*

We present the full algorithm for memory reduction.

Algorithm 1. (MEMORY REDUCTION)
Input: Infinite game $\Gamma = (G, \varphi)$
Output: Strategy automaton $\mathscr{A}_f$ for Player 0 from W_0

1. Establish a game simulation of Γ by a new game Γ' in which Player 0 has a positional winning strategy from W_0' (cf. Definition 1).
2. View Γ' as ω-automaton $\mathscr{A}$ (cf. Definition 2).
3. Reduce ω-automaton $\mathscr{A}$: Use a compatible equivalence relation $\approx$ on $S \times V$ to compute $\approx_S$ on S and construct the corresponding quotient game automaton $\mathscr{A}/_{\approx_S}$ (cf. Definitions 3,5).
4. View $\mathscr{A}/_{\approx_S}$ as automaton game $\Gamma'' = (G'', \varphi'')$ (cf. Definition 2).
5. Compute a positional winning strategy for Player 0 in Γ'' and from it construct the strategy automaton $\mathscr{A}_f$.

Algorithm 1 does not depend on the actual winning condition φ, but we need a suitable relation $\approx$ to execute step 3. Moreover, Theorem 1 is even valid if Γ' does not admit positional winning strategies.

4 Request-Response Games

In this section we apply the framework of the preceding section to request-response games. The first step is a game simulation by a Büchi game. The idea of this simulation is to memorize the set of open requests, and we use a marker (which is cyclically increased) to indicate which request is to be fulfilled next. Every time the marker is reset to value 1 we visit a final state.

Remark 2. Let $G = (V,E)$ be a game graph and $\Omega = \{(P_1,R_1),\dots,(P_k,R_k)\}$ a family of k pairs of subsets of V. Then the induced Request-Response game $\Gamma = (G,\Omega)$ is simulated by a Büchi game $\Gamma' = (G',F')$.

Proof. We define the game graph $G' = (V',E')$ and the set F' of final vertices as follows:

- $V' := 2^{\{1,\dots,k\}} \times \{1,\dots,k\} \times \{0,1\} \times V$
- $((P,i,b,v),(P',i',b',v')) \in E' :\Longleftrightarrow$
  - $P' = (P \cup \{i \mid v \in P_i\}) \setminus \{i \mid v \in R_i\}$
  - $i' = \begin{cases} i & \text{if } i \in P' \\ (i \bmod k)+1 & \text{otherwise} \end{cases}$
  - $b' = \begin{cases} 1 & \text{if } i = k \text{ and } i' = 1 \\ 0 & \text{otherwise} \end{cases}$
  - $(v,v') \in E$
- $F' := 2^{\{1,\dots,k\}} \times \{1,\dots,k\} \times \{1\} \times V$

□

It is easy to verify (see [12]) that the above construction satisfies Definition 1. In the sequel, we explain how to compute a compatible equivalence relation for Büchi game automata, using results on delayed simulation presented in [3].

4.1 Delayed Simulation for Büchi Automata

The delayed simulation game $\mathcal{G}_{de}(q_0,q_0')$ on a Büchi automaton $\mathscr{A}$ is played by two players, *Spoiler* and *Duplicator*, and starts at (q_0,q_0'), where q_0,q_0' are arbitrary states of $\mathscr{A}$. In the first round Spoiler chooses a transition $(q_0,a_0,q_1) \in \Delta$ and Duplicator answers by a transition $(q_0',a_0,q_1') \in \Delta$ with the same labeling. From the pair (q_1,q_1') the game proceeds with the second round, analogously, and so on, until infinity. This way, Spoiler and Duplicator build up two infinite paths $\rho = q_0q_1q_2\cdots$ and $\rho' = q_0'q_1'q_2'\cdots$, respectively. The play (ρ,ρ') is winning for Duplicator if and only if

$$\forall i(q_i \in F \Longrightarrow \exists j \geq i : q_j' \in F).$$

We say that q_0' *delayed simulates* q_0 if and only if Duplicator has a winning strategy in $\mathcal{G}_{de}(q_0,q_0')$ and denote this $q_0 \preceq_{de} q_0'$. Moreover, we say that q_0,q_0' delayed simulate each other, denoted $q_0 \simeq_{de} q_0'$, if and only if $q_0 \preceq_{de} q_0'$ and $q_0' \preceq_{de} q_0$. Quotienting with respect to $\simeq_{de}$ preserves the recognized language.

Lemma 1 ([3])**.** *Let $\mathscr{A}$ be a Büchi automaton. Then it holds $L(\mathscr{A}) = L(\mathscr{A}/_{\simeq_{de}})$.*

We extend delayed simulation to delayed *bisimulation* as follows: In each round of the play, Spoiler has a free choice whether to take the next transition in either ρ or ρ', and Duplicator must take the next transition in the other run, afterwards. The winning condition is modified as follows: If a final state is seen at position i of either run, then there must exist $j \geq i$ such that a final state is seen at position j in

the other run. If Duplicator wins the corresponding game, then we say that q_0, q_0' are delayed *bisimilar*, and denote this $q_0 \approx_{de} q_0'$.

We make use of the fact that, for deterministic Büchi automata, delayed simulation can be replaced by delayed bisimulation, and vice versa.

Remark 3. Let $\mathscr{A}$ be a deterministic Büchi automaton. Then, for all states q, q' of $\mathscr{A}$ it holds

$$q \simeq_{de} q' \iff q \approx_{de} q'.$$

Remark 3 follows immediately from the fact that the transitions taken in $\mathscr{A}$ are uniquely determined by the letters chosen by Spoiler. Hence, it makes no difference whether he chooses the next transition in ρ or ρ'. To compute $\approx_{de}$ we introduce *direct* bisimulation. It is defined as delayed bisimulation with the only difference of a modified winning condition in the corresponding game: The play $(q_0 q_1 q_2 \cdots, q_0' q_1' q_2' \cdots)$ is winning for Duplicator if and only if

$$\forall i (q_i \in F \iff q_i' \in F).$$

If Duplicator has a winning strategy in the direct bisimulation game, starting at (q_0, q_0'), then we say that q_0, q_0' are direct bisimilar, and we denote this $q_0 \approx_{di} q_0'$. To explain how delayed bisimulation and direct bisimulation are connected, we introduce the *closure* of $\mathscr{A}$, denoted $\mathrm{cl}(\mathscr{A})$. It has the set F' of final states, where we initially set $F' := F$ and iterate the following until a fixed point is reached:

If there exists $q \notin F'$ such that all successors of q are in F', then put q in F'.

The automata $\mathscr{A}$ and $\mathrm{cl}(\mathscr{A})$ are equivalent and, clearly, $\mathrm{cl}(\mathscr{A})$ is deterministic if $\mathscr{A}$ is deterministic. Moreover, note that $\mathrm{cl}(\mathscr{A})$ can be computed in time linear in $|\mathscr{A}|$. The following lemma completes the approach to state space reduction of (deterministic) Büchi automata.

Lemma 2 ([3])**.** *Let $\mathscr{A}$ be a Büchi automaton. For all states q, q' we have*

$$q \approx_{de} q' \text{ in } \mathscr{A} \iff q \approx_{di} q' \text{ in } \mathrm{cl}(\mathscr{A}).$$

For a given Büchi game automaton $\mathscr{A}$ we compute $\mathscr{A}/_{\approx_S}$ as follows: We compute the direct bisimulation relation $\approx_{di}$ in $\mathrm{cl}(\mathscr{A})$ which coincides with $\simeq_{de}$ in $\mathscr{A}$. Note that for a deterministic Büchi automaton the computation of $\approx_{di}$ is the same as block partitioning for standard DFA. Hence, the computation can be done in time $\mathscr{O}(n \log n)$, if n is the number of states of $\mathrm{cl}(\mathscr{A})$ and $|\Sigma|$ is assumed constant [9]. As a direct consequence, we get that $\approx_{di}$ is compatible with $\mathrm{cl}(\mathscr{A})$ and $\approx_{de}$ (hence, also $\simeq_{de}$) is compatible with $\mathscr{A}$. From $\simeq_{de}$, the relation $\approx_S$ is computed as given in Definition 4 where $\approx$ is replaced by $\simeq_{de}$. Finally, we can apply Theorem 1.

Remark 4. Let $\mathscr{A}$ be a Büchi game automaton and $\simeq_{de}$ the delayed simulation relation for $\mathscr{A}$. Then $\simeq_{de}$ is compatible with $\mathscr{A}$.

Corollary 1. *Let Γ be a Request-Response game and Γ' the corresponding Büchi game (cf. Remark 2). Further, let $\mathscr{A}$ be the game automaton of Γ' and $\simeq_{de}$ defined as above. Then Γ is simulated by the automaton game Γ'' of $\mathscr{A}/_{\approx_S}$.*

Let us briefly give the memory reduction algorithm for Request-Response games.

Algorithm 2. (MEMORY REDUCTION FOR REQUEST-RESPONSE GAMES)
Input: Request-Response game $\Gamma = (G, \Omega)$
Output: Strategy automaton $\mathscr{A}_f$ for Player 0 from W_0

1. Establish a game simulation of Γ by a Büchi game Γ' (cf. Remark 2).

2. View Γ' as Büchi game automaton $\mathscr{A}$ (cf. Definition 2).
3. Compute $\mathrm{cl}(\mathscr{A})$ (cf. page 52) and the direct bisimulation relation $\approx_{di}$ in $\mathrm{cl}(\mathscr{A})$. By Lemma 2 and Remark 3 it coincides with $\simeq_{de}$ in $\mathscr{A}$. From $\simeq_{de}$ compute $\approx_S$ (cf. Definition 4).
4. View $\mathscr{A}/_{\approx_S}$ as Büchi automaton game Γ'' (cf. Definition 2).
5. Compute a positional winning strategy for Player 0 in Γ'' and from it construct $\mathscr{A}_f$.

We have shown that the delayed simulation relation of a deterministic Büchi automaton can be computed in time $\mathscr{O}(n \cdot \log n)$, where n is the number of states and $|\Sigma|$ is assumed constant. Here, we get a complexity of $\mathscr{O}(n \cdot (\log n)^2)$, because we have $\mathscr{O}(\log n)$ input letters. The Büchi game Γ'' can be solved in time $\mathscr{O}(n^2 \cdot \log n)$. Hence, the total running time of Algorithm 2 is polynomial in $|\Gamma'|$.

4.2 An Example for Request-Response Games

In this subsection we compare the memory size of winning strategies (obtained by Algorithm 2) with the standard approach, where after the conversion of a request-response game into a Büchi game a winning strategy is directly computed (and then possibly minimized according to I/O-automata minimization). We present an example with an exponential gain in the memory size.

Consider the game graph G_k, which is shown in Figure 2 for $k = 3$. Let Ω be the following request-response winning condition:

$$\Omega_k = \{(P_0, R_0)\} \cup \{(P_1, R_1), (P_1', R_1'), \ldots, (P_k, R_k), (P_k', R_k')\}$$

A play proceeds as follows: From the initial vertex v, Player 1 takes k decisions activating either P_i or P_i' ($i = 1, \ldots, k$). At vertex w Player 0 takes over making k decisions himself. In vertex y all pairs from Ω are responded to. Hence, each (positional) strategy for Player 0 is winning.

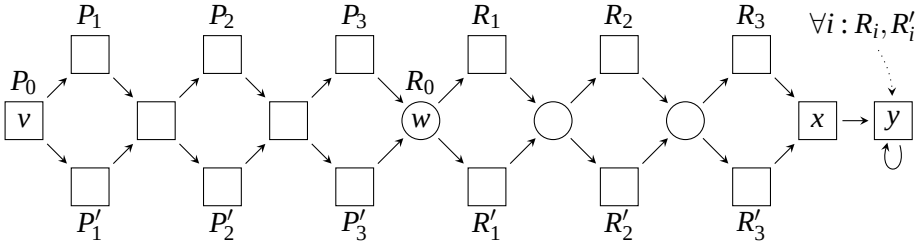

Figure 2: *Request-Response Game Graph G_3*

Theorem 2. *Let $\Gamma_k = (G_k, \Omega_k)$ be the Request-Response game from Figure 2 and let $\Gamma_k' = (G_k', F')$ be the Büchi game simulating Γ_k, constructed as in the proof of Remark 2. Then, Player 0 wins Γ_k from v such that the following hold:*

1. *The positional winning strategy f_k' for Player 0 from $(\varnothing, 1, 0, v)$ in Γ_k' yields a winning strategy f_k for Player 0 from v in Γ_k of size at least 2^k.*
2. *The reduced game graph G_k'' computed by Algorithm 2 has only one memory content.*

Proof. If Player 0 precisely mimics the decisions of Player 1, i.e., for all $i = 1, \ldots, k$ she moves to the R_i-vertex if and only if Player 1 has moved to the P_i-vertex before, then in the Büchi game a final vertex

is seen as soon as vertex y is visited for the first time. On the way from vertex w to vertex y, the counter in the second component of the memory is increased by one for $2k+1$ times: It starts with value 1 at vertex w and has value $2k+1$ when reaching vertex x; it is reset to value 1 when the play proceeds from vertex x to vertex y.

If Player 0 makes a mistake, i.e., there exists i such that she moves to the R_i-vertex if and only if Player 1 has moved to the P_i'-vertex, then a final vertex is reached several moves later than in the case where she plays "correctly". This is due to the fact that her false decision at the ith response avoids that the counter in the second component of the memory is increased.

By our remarks above, there is a unique shortest path from vertex w to vertex y which visits a final vertex in the Büchi game. It is the path which precisely mimics the path from vertex v to vertex w. Solving the Büchi game we obtain an attractor strategy, which means that a final vertex is assumed as soon as possible. Hence, the strategy chooses the "correct" path from vertex w to vertex y, and this strategy requires a memory of size at least 2^k because it needs to memorize each of the k decisions of Player 1.

Now, let us consider Algorithm 2. Let Γ' be the Büchi game computed by the game simulation from Remark 2, and let $s := (\varnothing, 1, 1, y) \in S \times V$. Once the play on G reaches vertex y, the set of active pairs is emptied, whereby on G' we reach a vertex of the form $(\varnothing, i, b, y)$, where $1 \leq i \leq k, b \in \mathbb{B}$. After at most k revisits to vertex y (on G) the value of i is reset to 1, which means that we reach vertex s (on G'). Moreover, vertex s is repeatedly visited every $2k+1$ moves, thereafter. Summing up, every infinite path on G' visits vertex s (infinitely often).

Consider the Büchi game automaton $\mathscr{A}$ of Γ'. By the remarks above, every infinite path in $\mathscr{A}$ (without q_{sink}) leads through s, even when starting at non-reachable states. Accordingly, all states in $S \times V$ are declared final in $\text{cl}(\mathscr{A})$. Thus, we obtain $(s_1, v) \approx_{di} (s_2, v)$ for all $s_1, s_2 \in S, v \in V$. Accordingly, all memory contents are equivalent, i.e. $S/_{\approx_S}$ is a singleton. □

5 Streett Games

A Streett winning condition is given by a family Ω of pairs of subsets of V:

$$\Omega = \{(E_1, F_1), \ldots, (E_k, F_k)\}$$

Player 0 wins if and only if, for each j, infinitely many visits to F_j imply infinitely many visits to E_j.

In a game simulation for Streett games by parity games we keep track of the order of the latest visits to the sets F_i, E_i ($i = 1, \ldots, k$). To do so, we use a data structure called *Index Appearance Record*, short IAR [5]. For $k \geq 1$, we denote $\mathscr{S}_k$ the symmetric group of $\{1, \ldots, k\}$, i.e., the set of all its permutations.

Remark 5. Let $G = (V, E)$ be a game graph and $\Omega = \{(E_1, F_1), \ldots, (E_k, F_k)\}$ a family of pairs of subsets of V. Then the induced Streett game $\Gamma = (G, \Omega)$ is simulated by a parity game $\Gamma' = (G', c')$.

Proof. Let $G' = (V', E')$ be defined as follows. As memory S we use the Index Appearance Record (IAR) of V:

$$S := \text{IAR}(V) = \{(i_1 \cdots i_k, e, f) \mid (i_1 \cdots i_k) \in \mathscr{S}_k, 1 \leq e, f \leq k\}$$

As initial memory content we choose $s_0 := (1 \cdots k, 1, 1)$. The transition relation E' is uniquely determined by E and Ω. We define:
$(((i_1 \cdots i_k, e, f), v), ((i_1' \cdots i_k', e', f'), v')) \in E' :\Longleftrightarrow$

1. $(v, v') \in E$

2. $(i'_1 \cdots i'_k)$ is obtained from $(i_1 \cdots i_k)$ by shifting all i_l with $v \in E_{i_l}$ to the left, $l \in \{1,\ldots,k\}$
3. e' is the maximal[1] $l \in \{1,\ldots,k\}$ such that $v \in E_{i_l}$
4. f' is the maximal $m \in \{1,\ldots,k\}$ such that $v \in F_{i'_m}$

The coloring $c' : \mathrm{IAR}(V) \times V \to \{1,\ldots,2k\}$ of V' is defined by:

$$c'((i_1 \cdots i_k, e, f), v) := \begin{cases} 2e & \text{if } e \geq f \\ 2f-1 & \text{if } e < f \end{cases}$$

□

We use the right-hand delayed simulation for alternating parity automata (introduced in [4]) to reduce parity game automata. Whereas for Büchi game automata the problem of computing delayed simulation can be reduced to the minimization problem for standard DFA (cf. Section 4.1), for parity game automata we have to solve the corresponding simulation game explicitly. It is described in [4], and we can use a simplified version of it. Firstly, a parity game automaton is not alternating which means that the first move of each round is made by Spoiler in the simulated automaton and the second move is made by Duplicator in the simulating automaton. Due to this fixed order of moving the pebbles we need less vertices in the simulation game graph. Secondly, a parity game automaton is deterministic. This means that the positions of the two pebbles and the update of the priority memory (see below) are uniquely determined by the letter chosen by Spoiler. Hence Duplicator's moves are predetermined by Spoiler's moves and, accordingly, all vertices in the simulation game graph belong to Spoiler. Let us define the simulation game in a formal way.

5.1 Right-hand Delayed Simulation for Parity Automata

We are given a parity game automaton $\mathscr{A} = ((S \times V) \cup \{q_0, q_{\text{sink}}\}, q_0, \delta, c', V_0)$ over V, where a run ρ of $\mathscr{A}$ is accepting if and only if the maximal color seen infinitely often in ρ is even. In [4] a min-parity condition is assumed. Hence, we have to redefine the coloring of $\mathscr{A}$ by $c' := k - c'$ for even $k \in \mathbb{N}$ large enough. We construct the simulation game $\mathscr{G}^{rh}_{de} = (G^{rh}_{de}, \varphi^{rh}_{de})$ as follows: The game graph $G^{rh}_{de} = (V^{rh}_{de}, E^{rh}_{de})$ has the set of vertices $V^{rh}_{de} = (S \times V) \times (S \times V) \times (c'(S \times V) \cup \{\checkmark\})$, where $c'(S \times V)$ denotes the set of colors assigned by the parity function c'. We set $V_{\text{Sp}} := V^{rh}_{de}$ (and $V_{\text{Du}} := \varnothing$). The edge relation $E^{rh}_{de} \subseteq V^{rh}_{de} \times V^{rh}_{de}$ is defined as follows:

$$\begin{gathered}(((s_1,v_1),(s_2,v_2),k),((s'_1,v'_1),(s'_2,v'_2),k')) \in E^{rh}_{de} :\Longleftrightarrow \\ ((s_1,v_1),(s'_1,v'_1)) \in E', ((s_2,v_2),(s'_2,v'_2)) \in E', v'_1 = v'_2 \text{ and} \\ k' = \mathrm{pm}(c'(s'_1,v'_1), c'(s'_2,v'_2), k)\end{gathered}$$

The *priority memory* update function $\mathrm{pm} : \mathbb{N} \times \mathbb{N} \times (\mathbb{N} \cup \{\checkmark\}) \to \mathbb{N} \cup \{\checkmark\}$ is defined as follows:

i. $\mathrm{pm}(i,j,\checkmark) = \min\{i,j\}$, if $i \prec j$
ii. $\mathrm{pm}(i,j,\checkmark) = \checkmark$, if $j \preceq i$
iii. $\mathrm{pm}(i,j,k) = \min\{i,j,k\}$, if $i \prec j$
iv. $\mathrm{pm}(i,j,k) = k$, if $j \preceq i$, i is odd and $i \leq k$, and j is odd or $k < j$
v. $\mathrm{pm}(i,j,k) = \checkmark$, if $j \preceq i$, j is even and $j \leq k$, and i is even or $k < i$
vi. $\mathrm{pm}(i,j,k) = \checkmark$, if i is odd, j is even, and both $i \leq k$ and $j \leq k$

[1] We assume w.l.o.g. that $E_k = F_k = V$ to have the pointers e' and f' well-defined.

vii. else $\mathrm{pm}(i,j,k) = k$

In the basic definition of the delayed simulation game in [4] the value of pm in case iv is set to ✓. It is also shown there that quotienting with respect to the obtained equivalence relation is not language-preserving. Hence, we use the slightly modified version from above, where in case iv the value of pm is set to k. The induced relation is defined on page 56 and preserves the recognized language. (This is also shown in [4].) The binary relation $\prec$ is the *reward order* on $\mathbb{N}$. For $m,n \in \mathbb{N}$, we define $m \preceq n$ if and only if

1. m is even and n is odd, or
2. m and n are both even and $m \leq n$, or
3. m and n are both odd and $n \leq m$.

This yields $0 \prec 2 \prec 4 \prec \ldots \prec 5 \prec 3 \prec 1$. If $m \prec n$ then we say that m is *better* than n, whereas terms like *minimum* and *smaller* refer to the standard relation $<$ on $\mathbb{N}$. We leave it up to the reader to verify that case vii of the definition of pm applies if and only if $j \preceq i, k < i$ and $k < j$.

A play ρ is winning for Duplicator if and only if the set

$$F := (S \times V) \times (S \times V) \times \{\checkmark\}$$

is visited infinitely often; this means that φ^{rh}_{de} is a Büchi condition. (Spoiler wins if and only if he can avoid ✓ from a certain point onwards.) We say that (s_2,v_2) *right-hand delayed simulates* (s_1,v_1), denoted $(s_1,v_1) \leq^{rh}_{de} (s_2,v_2)$, if and only if Duplicator has a winning strategy in $\mathscr{G}^{rh}_{de}$ from the initial game position $\mathrm{p_I}((s_1,v_1),(s_2,v_2))$ defined as follows. Let $i := c'(s_1,v_1)$ and $j := c'(s_2,v_2)$:

$$\mathrm{p_I}((s_1,v_1),(s_2,v_2)) := \begin{cases} ((s_1,v_1),(s_2,v_2),\min\{i,j\}) & \text{if } i \prec j \\ ((s_1,v_1),(s_2,v_2),\checkmark) & \text{otherwise} \end{cases}$$

In [4] it is shown that $\leq^{rh}_{de}$ is a preorder implying language containment, i.e., if $(s_1,v_1) \leq^{rh}_{de} (s_2,v_2)$ then $L(\mathscr{A}_{(s_1,v_1)}) \subseteq L(\mathscr{A}_{(s_2,v_2)})$. We define the corresponding equivalence relation $\approx^{rh}_{de}$ as

$$(s_1,v_1) \approx^{rh}_{de} (s_2,v_2) :\iff (s_1,v_1) \leq^{rh}_{de} (s_2,v_2) \text{ and } (s_2,v_2) \leq^{rh}_{de} (s_1,v_1).$$

Duplicator's winning region W_{Du} in $\mathscr{G}^{rh}_{de}$ determines $\leq^{rh}_{de}$, and from that we can compute $\approx^{rh}_{de}$. Note that we need to consider only the case where it holds $v_1 = v_2$ (cf. Definition 4).

5.2 Quotienting

The relation $\approx^{rh}_{de}$ is compatible with a parity game automaton. Item 1 of Definition 3 is verified by the upcoming lemma. Item 2 of Definition 3 follows from the fact that quotienting with respect to $\approx^{rh}_{de}$ is language-preserving, as is shown in [4].

Lemma 3. *Let $\mathscr{A}$ be a parity game automaton and $\approx^{rh}_{de}$ defined as in Section 5.1. Then, for all $s_1,s_2 \in S, v_1,v_2,v' \in V$ it holds:*

$$(s_1,v_1) \approx^{rh}_{de} (s_2,v_2) \Longrightarrow \delta((s_1,v_1),v') \approx^{rh}_{de} \delta((s_2,v_2),v')$$

Remark 6. Let $\mathscr{A}$ be a parity game automaton and $\approx^{rh}_{de}$ the right-hand delayed simulation relation for $\mathscr{A}$. Then $\approx^{rh}_{de}$ is compatible with $\mathscr{A}$. The quotient automaton $\mathscr{A}/_{\approx^{rh}_{de}}$ is defined in the natural way: $\delta/_{\approx^{rh}_{de}}([(s,v)],v') := [\delta((s,v),v')]$ and $c'/_{\approx^{rh}_{de}}([(s,v)]) := \min\{c'(s',v') \mid (s',v') \approx^{rh}_{de} (s,v)\}$; $\mathscr{A}/_{\approx^{rh}_{de}}$ is equivalent to $\mathscr{A}$. We refer to [4] for the details.

We compute $\approx_S$ from $\approx_{de}^{rh}$ (as in Definition 4) and express the acceptance condition $\psi/_{\approx_S}$ of $\mathscr{A}/_{\approx_S}$ in terms of a coloring $c'/_{\approx_S}$. To this end, let $s \in S, v \in V$ and define

$$c'/_{\approx_S}([s],v) := \min\{c'(s',v') \mid (s',v') \approx_{de}^{rh} (s,v)\},$$

and let q_0, q_{sink} inherit their color from $\mathscr{A}$. Since $s_1 \approx_S s_2$ implies $(s_1,v) \approx_{de}^{rh} (s_2,v)$ for all $v \in V$, the above definition of $c'/_{\approx_S}$ is independent of representatives. Note that $\mathscr{A}/_{\approx_S}$ is a game automaton. Essentially, the relation $\approx_S$ is a refinement of $\approx_{de}^{rh}$. Hence, the automaton $\mathscr{A}/_{\approx_S}$ is equivalent to $\mathscr{A}$.

Lemma 4. *Let $\mathscr{A}$ be a parity game automaton and $\mathscr{A}/_{\approx_S}$ the corresponding $\approx_S$-quotient with coloring $c'/_{\approx_S}$ (see above). Then $\mathscr{A}$ and $\mathscr{A}/_{\approx_S}$ are equivalent.*

Proof. We have to show $L(\mathscr{A}) = L(\mathscr{A}/_{\approx_S})$, where it suffices to show $L(\mathscr{A}/_{\approx_{de}^{rh}}) = L(\mathscr{A}/_{\approx_S})$. By Lemma 3 automaton $\mathscr{A}/_{\approx_{de}^{rh}}$ is deterministic, and by Definition 5 automaton $\mathscr{A}/_{\approx_S}$ is deterministic. For $\alpha \in V^\omega$, let ρ be the run of $\mathscr{A}/_{\approx_S}$ on α and ρ' be the corresponding run of $\mathscr{A}/_{\approx_{de}^{rh}}$ on α. The run ρ' is uniquely determined by the run ρ, because both $\mathscr{A}/_{\approx_{de}^{rh}}$ and $\mathscr{A}/_{\approx_S}$ are deterministic and $\approx_S$ is a refinement of $\approx_{de}^{rh}$. Moreover, ρ is accepting if and only if ρ' is accepting, because both runs have the same sequence of colors. Since both $\mathscr{A}/_{\approx_S}$ and $\mathscr{A}/_{\approx_{de}^{rh}}$ are deterministic, there is no other run on α, neither for $\mathscr{A}/_{\approx_S}$ nor for $\mathscr{A}/_{\approx_{de}^{rh}}$. Thus, α is accepted by $\mathscr{A}/_{\approx_S}$ if and only if it is accepted by $\mathscr{A}/_{\approx_{de}^{rh}}$. □

Our above results show that our algorithm for memory reduction is applicable to a Streett game Γ as follows: We simulate Γ by a parity game Γ' which is then transformed into a parity game automaton $\mathscr{A}$. For $\mathscr{A}$ we construct the right-hand delayed simulation game $\mathscr{G}_{de}^{rh}$ and solve it by standard techniques [5]. Duplicator's winning region in this game and Definition 4 uniquely determine $\approx_S$. The corresponding quotient automaton $\mathscr{A}/_{\approx_S}$ is a parity game automaton equivalent to $\mathscr{A}$, and we can transform it into a unique parity automaton game Γ''. By Theorem 1, Γ is simulated by Γ''.

Corollary 2. *Let Γ be a Streett game and Γ' the corresponding parity game (cf. Remark 5). Further, let $\mathscr{A}$ be the game automaton of Γ' and $\approx_{de}^{rh}$ defined as above. Then Γ is simulated by the automaton game Γ'' of $\mathscr{A}/_{\approx_S}$.*

This yields the following algorithm.

Algorithm 3. (MEMORY REDUCTION FOR STREETT GAMES)
Input: Streett game $\Gamma = (G, \Omega)$
Output: Strategy automaton $\mathscr{A}_f$ for Player 0 from W_0

1. Establish a game simulation of Γ by a parity game Γ' (cf. Remark 5).
2. View Γ' as parity game automaton $\mathscr{A}$ (cf. Definition 2); redefine the coloring of $\mathscr{A}$ as $c' := 2k - c'$.
3. Construct the delayed simulation game $\mathscr{G}_{de}^{rh}$ for $\mathscr{A}$ and solve it. From Duplicator's winning region compute $\approx_S$ (cf. Definition 4).
4. View $\mathscr{A}/_{\approx_S}$ as parity automaton game Γ'' (cf. Definition 2).
5. Compute a positional winning[2] strategy for Player 0 in Γ'' and from it construct $\mathscr{A}_f$.

At this point we have to mention an optional normalization, which may make the relation $\approx_{de}^{rh}$ larger. It is done before executing step 3. For each SCC C of $\mathscr{A}$ we iterate the following: While there exists $(s,v) \in C$ such that $c'(s,v) \geq 2$ and there exists no $(s',v') \in C$ such that $c'(s',v') = c'(s,v) - 1$ do $c'(s,v) := c'(s,v) - 2$. Clearly, this does not change the accepted language [4].

[2] Note that Γ'' is a min-parity game.

For the computation of $\approx_{de}^{rh}$ we need to solve the simulation game $\mathscr{G}_{de}^{rh}$. It is a Büchi game of size $\mathscr{O}(r^2 \cdot k)$ where r is the number of states of $\mathscr{A}$, and k is the number of colors assigned by c'. Since Büchi games are solvable in polynomial time measured in the size of the game graph, the overall running time of Algorithm 3 is polynomial in $|\Gamma'|$.

Note that the above technique can analogously be applied to Muller games (see [6]). The only difference is the game simulation in step 1. Muller games can also be simulated by parity games, but the needed memory depends on the number of vertices of the game graph G. Accordingly, we need to redefine $c' := 2|V| - c'$ at the end of step 2. The rest of the algorithm is the same as for Streett games, and we obtain a similar running time.

5.3 An Example for Streett Games

Let us show a result for the class of strong winning conditions, similar to the one above. We consider the Streett game in Example 1 (see below) and make particular assumptions on the winning strategy for Player 0 in the simulating parity game. More precisely, we demand that she behaves "optimal". This is meant in the sense that she continuously chooses those edges which globally guarantee the best colors she can enforce. (A color m is better than a color n if $n \prec m$, where $\prec$ is the reward order from page 56.)

Example 1. Let G_k be the graph shown in Figure 3 (for $k = 3$), and Ω_k the following Streett winning condition:

$$\Omega_k = \{(E_1,F_1),(E_{-1},F_{-1}),\ldots,(E_k,F_k),(E_{-k},F_{-k}),(V,V)\}$$

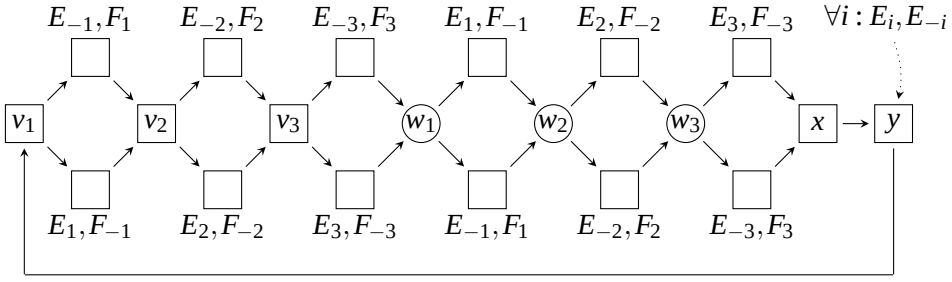

Figure 3: *Streett Game Graph* G_3

The game proceeds similarly to the one from Figure 2. The major difference is that vertex v_1 is visited infinitely often, naturally dividing each play into rounds. At the end of each round, i.e., when the play proceeds from vertex y to vertex v_1, the highest possible color $4k+2$ is seen in the parity game Γ'_k (simulating Γ_k). This is due to the fact that some index must be at the last position of the current IAR and, accordingly, the pointer e' has the value $2k+1$ (cf. page 55). Thus, each play satisfies the parity winning condition, and each (positional) strategy for Player 0 is winning.

Theorem 3. *Let $\Gamma_k = (G_k,\Omega_k)$ be the Streett game from Example 1 and let $\Gamma'_k = (G'_k,c_k)$ be the parity game simulating Γ_k (according to Remark 5), where $s_0 := ((1\cdots 2k+1),1,1)$ is the initial memory content. Then, Player 0 wins Γ_k from vertex v_1 such that the following hold:*

1. *Each positional winning strategy f'_k for Player 0 in Γ'_k from (s_0,v_1) with*

$$\{c_k(s',v') \mid ((s,v),(s',v')) \in E_{f'_k}{}^3\} \cap \{2n+1 \mid n \in \mathbb{N}\} = \varnothing$$

[3] $E_{f'_k}$ denotes the set of all edges determined by the positional strategy f'_k.

yields a winning strategy f_k for Player 0 in Γ_k from v_1 of size at least 2^k.

2. *The reduced game graph G''_k computed by Algorithm 3 has only one memory content.*

Proof. For simplicity, we assume $k = 3$; the proof is analogous for other values of k. First, we fix a convention on the entries in an IAR: Let the pair (V,V) be represented by V and every other pair by its unique index, i.e., (E_{-2},F_{-2}) is represented by -2; if index i has value $-j$, then let $-i := j$, for $1 \leq j \leq 3$.

The intersection in item 1 means that Player 0 chooses only edges leading into vertices of even color. To prove that item, note that the permutation reached (in Γ'_3) when vertex w_1 is reached (in Γ_3) is of the form

$$(V\, i_3 \cdots i_1\, p),$$

where $i_j \in \{j,-j\}$ (for $1 \leq j \leq 3$), and p is some permutation of the set $\{-i_1,-i_2,-i_3\}$. For example, let Player 1 at v_1 decide to move up, then again up, and then down; then the permutation is $(V\ 3\ {-2}\ {-1}\ p)$, because the indices -1, -2 and 3 are shifted to the second position one after another, and V stays at the front (cf. proof of Remark 5). Moreover, p is a permutation of $\{1,2,-3\}$, i.e., of the set of all indices i for which Player 1 has moved to F_i, recently.

If Player 0 moves upwards at w_1, i.e., she mimics Player 1's behavior at vertex v_1, then the permutation shortly becomes $(V\ 1\ 3\ {-2}\ {-1}\ p')$, and p' is either $(2\ {-3})$ or $({-3}\ 2)$. That means e' is assigned the value 5, 6 or 7.[4] Simultaneously, f' gets the value 5 because F_{-1} is visited and -1 is at the fifth position in the new permutation. Accordingly, it holds $e' \geq f'$, which means that we see color 10, 12 or 14.

Conversely, if Player 0 moves downwards at w_1, then the permutation becomes $(V\ {-1}\ 3\ {-2}\ p'')$, with $p'' = p$. Hence, e' is assigned 4 because index -1 comes from the fourth position. Moreover, index 1 is located somewhere in p'', i.e., at position 5, 6 or 7. Thus, we have $f' \geq 5 > 4 = e'$; accordingly, we see an odd color, either 9, 11 or 13.

Making an analogous observation at vertices w_2, w_3, we can deduce the following: If Player 0 mimics Player 1's behavior then she visits an even color, say l; if she makes the "wrong" move then a vertex of odd color less than l is seen. Let the latter situation be called an *error* and note that Player 0 can play errorless by memorizing Player 1's decisions. By an argument analogous to that in the proof of Theorem 2, implementation of an errorless winning strategy requires a memory of size at least 2^k.

To see item 2, let us consider Algorithm 3. First, note that the coloring c_k is redefined as $c_k := 4k+2-c_k$ (in step 2); accordingly, we are dealing with a min-parity condition from now on. Every play on G'_k must traverse an edge $((s_1,y),(s_2,v))$ infinitely often, for some IARs s_1, s_2. Thereby, a vertex with the smallest possible color 0 is visited, because there must be $1 \leq j \leq k$ such that index j or $-j$ is at the last position of the IAR s_1. Thus, in the simulation game the priority memory is reset to ✓ and a final vertex is visited, infinitely often. Accordingly, Duplicator has a winning strategy from each vertex in the simulation game graph. Summing up, all states (having the same V-component) in the parity game automaton $\mathscr{A}$ of Γ'_k are $\approx^{rh}_{de}$-equivalent, which means that all memory contents are declared $\approx_S$-equivalent. Thus, we obtain a reduced memory of size one. □

6 Conclusion

We have presented a method that reduces the memory for implementing winning strategies in Request-Response and Streett games. The key idea is to view the result of a game simulation as an ω-automaton

[4] Note that the definition of e' refers to the old permutation, and index 1 came from position 5, 6 or 7 (cf. page 55).

whose state space contains the memory to solve the given game. This state space is reduced via the notion of delayed simulation (cf. [3, 4]). The reduction is carried out only on the set of memory contents, where two memory contents are considered equivalent if, from them, Player 0 wins exactly the same plays. In our setting, delayed simulation can be computed in time $\mathcal{O}(n \cdot (\log n)^2)$ and $\mathcal{O}(n^2 \cdot k)$ for Büchi and parity game automata, respectively, where n is the number of states of the game automaton and k the number of colors (in the parity game automaton). In both cases our algorithm has a running time polynomial in the size of the simulating game Γ'.

Acknowledgment

The authors thank Wolfgang Thomas for his advice.

References

[1] J. Richard Büchi & Lawrence H. Landweber (1969): *Solving sequential conditions by finite-state strategies*. *Transactions of the AMS* 138, pp. 295–311, doi:10.2307/1994916.

[2] Stefan Dziembowski, Marcin Jurdziński & Igor Walukiewicz (1997): *How Much Memory is Needed to Win Infinite Games?* In: *Proceedings of the 12th LICS*. IEEE, Washington - Brussels - Tokyo, pp. 99–110, doi:10.1109/LICS.1997.614939.

[3] Kousha Etessami, Thomas Wilke & Rebecca A. Schuller (2005): *Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata*. *SIAM Journal on Computing* 34(5), pp. 1159–1175, doi:10.1137/S0097539703420675.

[4] Carsten Fritz & Thomas Wilke (2006): *Simulation Relations for Alternating Parity Automata and Parity Games*. In: *Proceedings of the 10th DLT*. *LNCS* 4036, Springer, pp. 59–70, doi:10.1007/11779148_7.

[5] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata, Logics and Infinite Games*. *LNCS* 2500, Springer, doi:10.1007/3-540-36387-4. Available at http://link.springer.de/link/service/series/0558/papers/2500/.

[6] Yuri Gurevich & Leo Harrington (1982): *Trees, Automata and Games*. In: *Proceedings of the 14th STOC*. San Francisco, CA., pp. 60–65, doi:10.1145/800070.802177.

[7] Michael Holtmann & Christof Löding (2007): *Memory Reduction for Strategies in Infinite Games*. In Jan Holub & Jan Zdárek, editors: *CIAA*. *LNCS* 4783, Springer, pp. 253–264, doi:10.1007/978-3-540-76336-9.

[8] Christof Löding (2001): *Efficient Minimization of Deterministic Weak ω-Automata*. *IPL* 79, pp. 105–109, doi:10.1016/S0020-0190(00)00183-6.

[9] Robert Paige & Robert E. Tarjan (1987): *Three partition refinement algorithms*. *SIAM J. Comput.* 16(6), pp. 973–989, doi:10.1137/0216062.

[10] Wolfgang Thomas (1995): *On the synthesis of strategies in infinite games*. In: *Proceedings of the 12th STACS*. *LNCS* 900, Springer, Munich, Germany, pp. 1–13, doi:10.1007/3-540-59042-0_57.

[11] Wolfgang Thomas (1997): *Languages, Automata and Logic*. In A. Salomaa & G. Rozenberg, editors: *Handbook of Formal Languages*. 3, Beyond Words, Springer, Berlin.

[12] Nico Wallmeier, Patrick Hütten & Wolfgang Thomas (2003): *Symbolic Synthesis of Finite-State Controllers for Request-Response Specifications*. In: *Proceedings of the 8th CIAA*. *LNCS* 2759, Springer, pp. 11–22, doi:10.1007/3-540-45089-0_3.

[13] Igor Walukiewicz (2004): *A Landscape with Games in the Background*. In: *Proceedings of the 19th LICS*. IEEE Computer Society, pp. 356–366, doi:10.1109/LICS.2004.4. Available at http://csdl.computer.org/comp/proceedings/lics/2004/2192/00/21920356abs.htm.