Memory Reduction for Strategies in Infinite Games

Michael Holtmann and Christof Löding

RWTH Aachen, Lehrstuhl für Informatik 7, 52056 Aachen, Germany
{holtmann, loeding}@i7.informatik.rwth-aachen.de

Abstract. We deal with the problem of reducing the memory necessary for implementing winning strategies in infinite games. We present an algorithm that is based on the notion of game reduction. The key idea of a game reduction is to reduce the problem of computing a solution for a given game to the problem of computing a solution for a new game which has an extended game graph but a simpler winning condition. The new game graph contains the memory to solve the original game. Our algorithm computes an equivalence relation on the vertices of the extended game graph and from that deduces equivalent memory contents. We apply our algorithm to Request-Response and Staiger-Wagner games where in both cases we obtain a running time polynomial in the size of the extended game graph. We compare our method to the technique of minimising strategy automata and present an example for which our approach yields a substantially better result.

1 Introduction

Infinite games constitute a powerful tool for synthesis and verification of reactive systems such as protocols and controllers (cf. [Wal04]). We consider the case of two players, Player 0 corresponds to the system and Player 1 to the environment. The system is represented by a finite directed graph $G = (Q, E)$ with each vertex belonging to either player. The players move in alternation building up a play ρ. After infinitely many steps Player 0 is declared the winner if ρ satisfies the winning condition φ (modelling the requirements demanded from the system). A solution to an infinite game is indicated by the sets of vertices from where the players can win, and corresponding winning strategies. Accordingly, solving an infinite game amounts to constructing controller programmes that realise certain specifications.

A Muller winning condition is given by a family $\mathcal{F} = \{F_1, \ldots, F_k\}$ with $F_i \subseteq Q$ (cf. [Mul63]). A play is won by Player 0 if the set of vertices visited infinitely often is one of the F_i. Büchi and Landweber showed that the problem of finding a solution to a Muller game is decidable (cf. [BL69]). Gurevich and Harrington showed that in Muller games there exist winning strategies for both players that need only finite memory (cf. [GH82]). The amount of memory needed to realise a specific strategy corresponds to the size of the respective controller programme, e.g. a finite automaton with output. Hence, the problem of memory reduction has been intensely investigated. Whereas some winning conditions can be solved by positional strategies, others require an exponential memory (cf. [Tho95],[Zie98]). In [DJW97] it is shown that there exist Muller games requiring a memory of size at least $n!$ if $\mathcal{O}(n)$ is the size of the game graph. A similar result is shown for Streett games in [Hor05]. There, the lower bound on the size of the needed memory is $k!$ where k is the number of Streett pairs.

Even if worst case lower bounds exist for specific games the average case requires less memory. One approach to memory reduction is to compute a strategy and then minimise the corresponding automaton. This procedure may yield a result of arbitrary size depending on the computed strategy. As a remedy, we present an algorithm which is applied before computing a strategy. It is based on the concept of game reduction, i.e. simulation of a winning condition by a simpler one at the expense of the size of the game graph. The idea is to reduce the size of the extended game graph before computing a winning strategy. Basically, this leads to simplifying the winning strategies in the original game. For reducing the game graph we proceed via a transformation to ω-automata and introduce a notion of equivalent memory contents.

Generally, our algorithm is applicable to all common winning conditions but we have implemented it only for Request-Response and Staiger-Wagner games. In a Request-Response game we are given a set $\mathcal{F} = \{(P_1, R_1), \ldots, (P_k, R_k)\}$ with $P_i, R_i \subseteq Q$. Player 0 wins if each visit to the set P_i is eventually followed by a visit to the set R_i. This type of game can be reduced to a Büchi game using a memory of size $2^k \cdot k$ which can be shown to be asymptotically optimal (cf. [WHT03]). In a Staiger-Wagner game we are given a set $\mathcal{F} = \{F_1, \ldots, F_k\}$ with $F_i \subseteq Q$, and Player 0 wins if the vertices visited at least once form one of the sets F_i. Staiger-Wagner games can be solved by a reduction to weak parity games and require a memory exponential in the size of the game graph (cf. [Hol07]).

In the next section we introduce the basic notions on infinite games and the term game reduction. After that we present the two approaches to memory reduction and compare them to each other: We present a family of games where the game reduction algorithm produces a memory of exponential size which is then reduced to constant size by our new algorithm. The minimisation algorithm for finite automata with output fails to do so because the algorithm for solving the game yields a too complicated strategy whose minimal automaton is of exponential size. We conclude by giving some remarks on the implementation of our algorithm.

This article is based on [Hol07], and any further details can be found there.

2 Preliminaries

We assume that the reader is familiar with the basic theory of ω-automata, as e.g. described in [Tho96], and restrict this introduction to infinite games. An *infinite game* $\Gamma = (G, \varphi)$ is played by two players on a finite directed graph $G = (Q, E)$ with no dead ends. The set $Q = Q_0 \mathbin{\dot\cup} Q_1$ is a disjoint union of Player 0 and Player 1 vertices. A *play* is a sequence $\rho = q_0 q_1 q_2 \ldots \in Q^\omega$ such that $(q_i, q_{i+1}) \in E$ for every $i \in \mathbb{N}$. The *winning condition* $\varphi \subseteq Q^\omega$ denotes the set of plays winning for Player 0.

A *strategy* for Player 0 is a partial function $f : Q^*Q_0 \dashrightarrow Q$ that assigns to any play prefix $q_0 \dots q_i$ with $q_i \in Q_0$ a state q_{i+1} such that $(q_i, q_{i+1}) \in E$. The function f is called a *winning strategy for Player 0 from* q if any play ρ starting in vertex q that is played according to f is won by Player 0. The *winning region* W_0 of Player 0 is the set of all vertices from where Player 0 has a winning strategy. A *strategy automaton* for Player 0 is a Mealy automaton $\mathcal{A} = (S, Q, s_0, \sigma, \tau)$ with a memory update function $\sigma : S \times Q \to S$ and a transition choice function $\tau : S \times Q_0 \to Q$. The strategy $f_\mathcal{A}$ implemented by $\mathcal{A}$ is defined as $f_\mathcal{A}(q_0 \dots q_i) := \tau(\sigma^*(s_0, q_0 \dots q_{i-1}), q_i)$ for $q_i \in Q_0$. A strategy f is called *positional* if it can be implemented by a strategy automaton with one state. The *size* of a strategy is the minimal number of states among all automata implementing this strategy. By a *solution* to an infinite game we mean the winning regions of the two players and corresponding winning strategies.

We now briefly introduce the winning conditions that are relevant for us. In a Büchi game we are given a designated set $F \subseteq Q$ and Player 0 wins if and only if the set of vertices visited infinitely often contains a state from F. Büchi games can be solved by positional strategies. In a weak parity game we are given a colouring $c : Q \to \{0, \dots, k\}$ with $k \in \mathbb{N}$. Player 0 wins ρ if and only if the maximal colour occurring in ρ is even. Weak parity games can be solved by a conventional algorithm where we obtain positional winning strategies for both players (cf. [Cha06]). We perform a backward breadth-first search through the game graph and compute for each even (resp. odd) colour the sets of vertices from where Player 0 (resp. Player 1) can win by reaching a vertex of this colour. Since each vertex can be removed after it has been visited the overall running time of the algorithm is in $\mathcal{O}(|E|)$.

There are more complex winning conditions which are not solvable by positional strategies. In a Request-Response game we are given a family of pairs $\mathcal{F} = \{(P_1, R_1), \dots, (P_k, R_k)\}$ and Player 0 wins if and only if for all $i = 1, \dots, k$ any visit to the set P_i is eventually followed by a visit to the set R_i. In a Staiger-Wagner game we are given a family $\mathcal{F} = \{F_1, \dots, F_k\}$ and Player 0 wins if and only if the set of visited vertices is one of the sets in $\mathcal{F}$ (cf. weak Muller games in [Tho02]). To solve these games we introduce the notion of game reduction: From the given game graph we construct a new game graph extended by a memory component, such that on this new game graph a simpler winning condition suffices to simulate the original game. The new edge relation comprises the memory update. Usually, the extended game graph gets exponentially large in the size of the original game graph.

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

1. $Q' = S \times Q$ for a finite memory set S (and $q \in Q_i \iff (s, q) \in Q'_i$)
2. Every play ρ of Γ is transformed into a unique play ρ' of Γ' by
   (a) $\exists s_0 \in S, \forall q \in Q$: $\rho(0) = q \Longrightarrow \rho'(0) = (s_0, q)$

(b) Let $(s,q) \in Q'$:
   i. $(q,q') \in E \Longrightarrow \exists s' \in S : ((s,q),(s',q')) \in E'$
   ii. $((s,q),(s_1,q_1)) \in E', ((s,q),(s_2,q_2)) \in E' \Longrightarrow s_1 = s_2$
(c) $((s,q),(s',q')) \in E' \Longrightarrow (q,q') \in E$
3. ρ is winning for Player 0 in $\Gamma \iff \rho'$ is winning for Player 0 in Γ'

From a positional winning strategy for Player 0 in Γ' we can directly construct a strategy automaton $\mathcal{A}$ implementing a winning strategy for Player 0 in Γ in the following way: The transition structure of G' yields the memory update function σ, and any positional winning strategy for Player 0 in Γ' yields a feasible output function τ for the automaton $\mathcal{A}$.

We now give a game reduction algorithm from Staiger-Wagner to weak parity games. As memory S we use the powerset of Q, i.e. $S = 2^Q$. The idea is to start with the empty set ($s_0 := \emptyset$) and accumulate the visited states in the memory. Note that in our definition of game reduction the memory update only depends on the source vertex of a transition (cf. item 2.(b).ii). This means for any $s \in S$ and any transition (q,q') in G we get a transition $((s,q),(s \cup \{q\},q'))$ in G' (and no other transitions). The colouring c for the weak parity condition is defined as follows:

$$c((R,q)) := \begin{cases} 2 \cdot |R \cup \{q\}| - 1 & \text{, if } R \cup \{q\} \notin \mathcal{F} \\ 2 \cdot |R \cup \{q\}| & \text{, if } R \cup \{q\} \in \mathcal{F} \end{cases}$$

With these definitions of S, E' and c one can easily verify that the properties from Def. 1 are satisfied. For a game reduction from Request-Response to Büchi games we refer to [WHT03].

3 Reduction of Strategy Automata

3.1 Minimisation of DFA with Output

Our aim is to present an algorithm for reducing the needed memory. One approach could be to compute a strategy and then minimise the corresponding automaton, i.e. a DFA with output (cf. [Hop71]). Two states are considered equivalent if taking either of them as initial state the same output functions are computed. This technique has a certain drawback because it depends on the implemented strategy. Consider the Staiger-Wagner game in the left part of Fig. 1 with $\mathcal{F} = \{\{0,1\},\{0,2\},\{0,1,2,3\}\}$. (Player 0 vertices are drawn as circles.)

Player 0 has the winning region $W_0 = \{0,1\}$ but does not have any positional winning strategy. If vertex 1 is visited then Player 0 has to move from vertex 2 to vertex 3. Otherwise, he has to stay at vertex 2 forever. To implement this winning strategy we need a memory of size two. Consider the strategy automaton $\mathcal{A}_n$ from the right part of Fig. 1 which, of course, is not the automaton obtained by a standard algorithm but only used to illustrate the problem. $\mathcal{A}_n$ implements the following strategy f_n: If vertex 1 is visited then stay in vertex 2 for exactly n times, then move on to vertex 3. If vertex 1 is not visited then stay in vertex 2. Obviously, for any $n \in \mathbb{N}$ this strategy is winning for Player 0 from W_0. Note

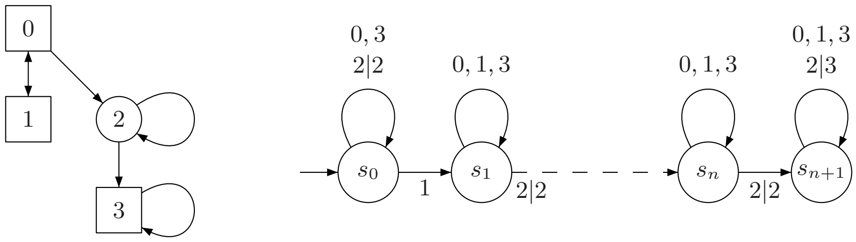

Fig. 1. *Simple Staiger-Wagner game*

that $\mathcal{A}_n$ is minimal w.r.t. the implemented strategy because we need the states $s_1, \dots, s_n$ to count the number of revisits to vertex 2. If we assume a large value for n then this is very unsatisfying. In Sect. 3.3 we provide a family of games for which a standard algorithm computes a complicated winning strategy although a simple one exists.

A second argument against the above minimisation technique is the fact that numerous transitions in a strategy automaton can be irrelevant for the definition of the implemented strategy. The general reason for this is that a strategy is a partial function. For a further discussion on problems arising in the minimisation of partially specified Mealy automata see [Koh70].

3.2 Reduction of Game Graphs

We present an algorithm which is independent of the winning strategies a player has in a specific game. The idea is to modify the output of the game reduction before constructing the strategy automaton. More precisely, we reduce the size of G' by computing an equivalence relation on S. We view Γ' as an ω-automaton accepting the winning plays for Player 0 in Γ and compute an equivalence relation $\approx$ on $S \times Q$. From $\approx$ we derive the equivalence relation $\approx_S$ on S. As the new memory we obtain the set $S' := S/_{\approx_S}$. Our algorithm is illustrated in Fig. 2.

Definition 2. Let $\Gamma = (G, \varphi)$ and $\Gamma' = (G', \varphi')$ be two infinite games and $\Gamma \leq \Gamma'$. We define the (deterministic) *game automaton* $\mathcal{A} = ((S \times Q) \,\dot\cup\, \{q_0, q_{\text{sink}}\}, Q_0, Q, q_0, \delta, \psi)$ where $Q_0 \subseteq Q$ is used to keep the information on the partition of the vertices. Basically, δ is adopted from E' and a transition is labelled by the Q-component of its target state. For $q' \in Q$ we set $\delta(q_0, q') := (s_0, q')$ and $\delta(q_{\text{sink}}, q') := q_{\text{sink}}$. For $s \in S, q, q' \in Q$ with $(q, q') \notin E$ we set $\delta((s, q), q') := q_{\text{sink}}$. The acceptance condition ψ is defined on an abstract level: A run $q_0\rho'$ of $\mathcal{A}$ is defined to be accepting if and only if ρ' is a winning play for Player 0 in Γ'. Constructing an infinite game from a given game automaton works the other way round. (For that we need Q_0.) We call this an *automaton game*.

Our idea is to reduce the game automaton in such a way that the properties of a game reduction are preserved. This primarily requires two things. First, to

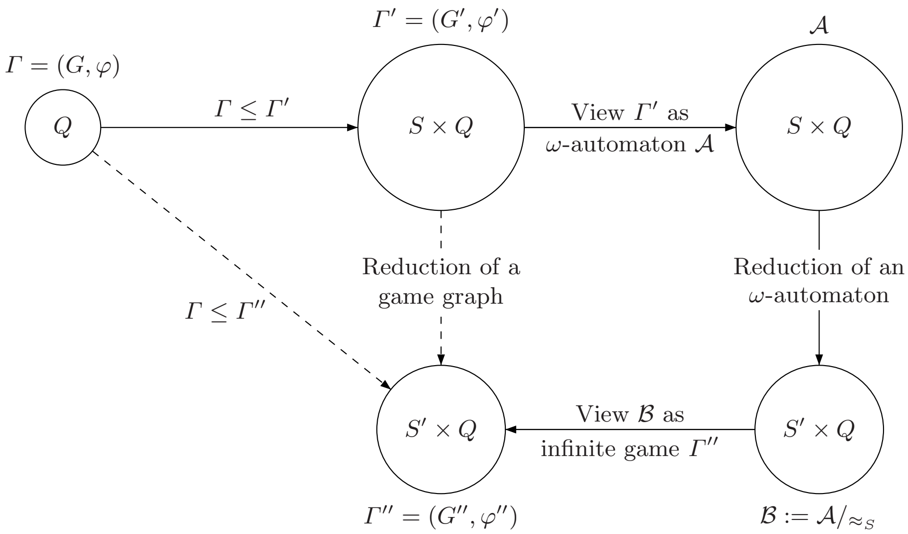

Fig. 2. *Algorithm for Memory Reduction*

retain item 1. from Def. 1 we are only allowed to modify the set S even if we proceed via an equivalence relation on the set $S \times Q$. Second, to achieve item 3. we have to preserve the recognised language. To be able to define the quotient automaton in a natural way we require the following structural properties for $\approx$.

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

1. For all $s_1, s_2 \in S, q, q' \in Q$:
   $(s_1, q) \approx (s_2, q) \Longrightarrow \delta((s_1, q), q') \approx \delta((s_2, q), q')$
2. Let ρ and ρ' be two runs in $\mathcal{A}$ starting at arbitrary states such that $\rho(i) \approx \rho'(i)$ for all $i \geq 0$. Then ρ is accepting if and only if ρ' is accepting.

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

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

$$s_1 \approx_S s_2 :\Longleftrightarrow \forall q \in Q : (s_1, q) \approx (s_2, q)$$

For $s \in S$, $[s]$ denotes the equivalence class of s w.r.t. $\approx_S$. Given $s_1 \approx_S s_2$ and $(q, q') \in E$, let $(s'_i, q') := \delta((s_i, q), q')$ for $i = 1, 2$. According to Def. 3 we know that $(s'_1, q') \approx (s'_2, q')$ holds. However, $s'_1 \approx_S s'_2$ does not hold necessarily. To get the q'-successor of $([s_1], q)$ 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* $\mathcal{A}/_{\approx_S} = ((S/_{\approx_S} \times Q) \dot\cup \{q_0, q_{\text{sink}}\}, Q_0, Q, q_0, \delta/_{\approx_S}, \psi/_{\approx_S})$. Given $([s], q) \in S/_{\approx_S} \times Q$ and $(q, q') \in E$ we define

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

where

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

The rest of $\delta/_{\approx_S}$ is defined analogously. Let $\rho = q_0([s_1], q_1)([s_2], q_2)\ldots$ be a run of $\mathcal{A}/_{\approx_S}$. We define ρ to be accepting if and only if the run $\rho' = q_0(s'_1, q_1)(s'_2, q_2)\ldots$ of $\mathcal{A}$ (which is uniquely determined by ρ) is accepting.

The run ρ' is uniquely determined by ρ because both $\mathcal{A}$ and $\mathcal{A}/_{\approx_S}$ are deterministic. The acceptance condition for $\mathcal{A}/_{\approx_S}$ immediately implies $L(\mathcal{A}) = L(\mathcal{A}/_{\approx_S})$. For reducing Büchi and weak parity game automata there exist equivalence relations which are computable efficiently (cf. Sect. 4). Moreover, in [Hol07] we show that each of these relations satisfies Def. 3 and that the respective quotient according to Def. 5 can be defined with the same type of acceptance condition. The claim of the following theorem is that the automaton game Γ'' of $\mathcal{A}/_{\approx_S}$ has the same structural properties as Γ', i.e. Γ is reducible to Γ'' (indicated by the left dashed arrow in Fig. 2).

Theorem 6. *Let $\Gamma = (G, \varphi)$ and $\Gamma' = (G', \varphi')$ be infinite games and Γ be reducible to Γ'. Let $\mathcal{A}$ be the game automaton of Γ' and $\approx$ a compatible equivalence relation on $S \times Q$. Then Γ is reducible to the unique automaton game Γ'' of $\mathcal{A}/_{\approx_S}$.*

Proof. We do not give the full proof here but refer to [Hol07] for the details. From Def. 5 it follows that $\mathcal{A}/_{\approx_S}$ has the state set $(S' \times Q) \dot\cup \{q_0, q_{\text{sink}}\}$ for some finite set S'. Hence, $\mathcal{A}/_{\approx_S}$ is a game automaton and we can transform it into a unique automaton game $\Gamma'' = (G'', \varphi'')$.

To prove that Γ is reducible to Γ'' we verify items 1. to 3. from Def. 1. Item 1. is immediately satisfied by our remarks above. As the initial memory content from item 2.(a) we choose the equivalence class $[s_0]$ of s_0 and obtain as initial game positions the set $\{([s_0], q) \mid q \in Q\}$. To show that the edge relation E transfers to E'' as required in items 2.(b).i and 2.(c) we argue by simple implications using Defs. 2 and 5 and the fact that Γ is reducible to Γ'. Since edge relation E', hence also δ, has the required properties and each transition in $\mathcal{A}/_{\approx_S}$ corresponds to a unique set of transitions in $\mathcal{A}$ we get that E'' also has the required properties. For item 2.(b).ii we show that the uniqueness of the memory update in G'' follows from the uniqueness of the memory update in G'. The proof is accomplished by contraposition.

What remains to be shown is that Player 0 wins a play ρ of Γ if and only if he wins the corresponding play ρ'' of Γ'' (item 3.). This is a simple consequence of the fact that automata $\mathcal{A}$ and $\mathcal{A}/_{\approx_S}$ are equivalent. $\square$

Since $S' = S/_{\approx_S}$ we get that $|S'| \leq |S|$. Of course, we want that $|S'| < |S|$. In the latter case the strategy automaton extracted from the game reduction $\mathit{\Gamma} \leq \mathit{\Gamma}''$ has less states than the one extracted from the game reduction $\mathit{\Gamma} \leq \mathit{\Gamma}'$. We now present the full memory reduction algorithm from Fig. 2 on page 6.

Algorithm 7. (Memory Reduction)
Input: Infinite game $\mathit{\Gamma} = (G, \varphi)$
Output: Strategy automaton $\mathcal{A}_f$ for Player 0 from W_0

1. Establish a game reduction from $\mathit{\Gamma}$ to a new game $\mathit{\Gamma}'$ in which Player 0 has a positional winning strategy from W_0' (cf. Def. 1).
2. View $\mathit{\Gamma}'$ as ω-automaton $\mathcal{A}$ (cf. Def. 2).
3. Reduce ω-automaton $\mathcal{A}$: Use a compatible equivalence relation $\approx$ on $S \times Q$ to compute $\approx_S$ on S and construct the corresponding quotient game automaton $\mathcal{B} := \mathcal{A}/_{\approx_S}$ (cf. Defs. 3,5).
4. Transform $\mathcal{B}$ into the unique automaton game $\mathit{\Gamma}'' = (G'', \varphi'')$ (cf. Def. 2).
5. Compute a positional winning strategy for Player 0 in $\mathit{\Gamma}''$ and from it construct the strategy automaton $\mathcal{A}_f$.

Note that Alg. 7 does not depend on the actual winning condition φ but that we only need a suitable relation $\approx$ to execute step 3. Moreover, Thm. 6 is even valid if $\mathit{\Gamma}'$ does not admit positional winning strategies.

3.3 Comparison of the Two Approaches

In this section we present a family of games where our new approach yields a substantial reduction of the used memory. Consider the Staiger-Wagner game $\mathit{\Gamma}_n = (G_n, \varphi_n)$ with game graph G_n from Fig. 3 and φ_n determined by the set

$$\mathcal{F}_n = \{U \mid U \subseteq \{v, u_1, \ldots, u_n\}, v \in U\} \cup \{R \mid R \supseteq \{x, y\}\}.$$

Accordingly, Player 0 wins if the play remains within $\{v, u_1, \ldots, u_n\}$ or reaches both x and y. Reducing $\mathit{\Gamma}_n$ to the weak parity game $\mathit{\Gamma}_n' = (G_n', \varphi_n')$ we obtain exponentially many reachable memory contents in G_n'. A memory content s is reachable if there exist $q, q' \in Q$ such that (s, q') is reachable from (s_0, q). If we solve $\mathit{\Gamma}_n'$ with the algorithm from [Cha06] then this yields an exponential winning strategy for Player 0 in $\mathit{\Gamma}_n$. If, however, we use our new algorithm to reduce G_n' before solving the game then we obtain a winning strategy of constant size.

Lemma 8. *Let $\mathit{\Gamma}_n = (G_n, \varphi_n)$ be defined as above and let $\mathit{\Gamma}_n' = (G_n', \varphi_n')$ be the weak parity game to which $\mathit{\Gamma}_n$ is reduced to (as described on page 4). Then Player 0 wins $\mathit{\Gamma}_n$ from vertex v such that the following hold:*

1. *The positional winning strategy f_n' for Player 0 in $\mathit{\Gamma}_n'$ from $(\emptyset, v)$ computed by the algorithm from [Cha06] yields a winning strategy f_n of exponential size for Player 0 in $\mathit{\Gamma}_n$ from v.*
2. *The reduced game graph G_n'' computed by Alg. 7 has constantly many memory contents.*

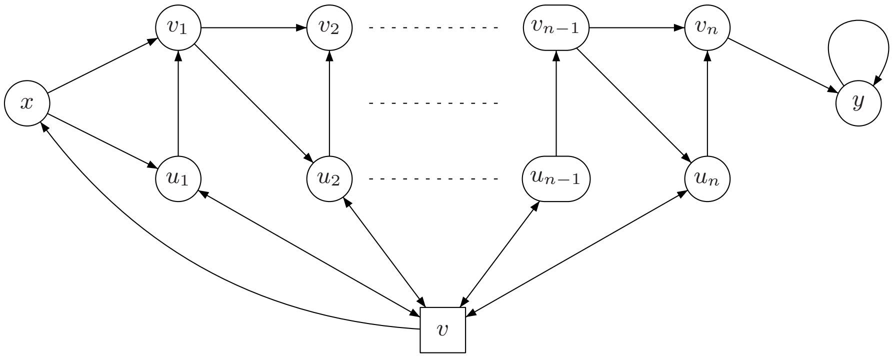

Fig. 3. *Staiger-Wagner game graph G_n*

Proof. We assume a play starting in vertex v and Player 1 eventually moving to vertex x. Thus, a play is of the form $\rho = v\rho_1 x\rho_2$. For abbreviation we set $U_v := \{U \mid U \subseteq \{v, u_1, \ldots, u_n\}, v \in U\}$. If we apply the game reduction algorithm from page 4 to Γ_n then we obtain a weak parity game Γ'_n where the game graph G'_n contains as memory the set of vertices visited in ρ. The algorithm for solving Γ'_n computes sets A_i from where a player can force a win by reaching the set $C_i := c^{-1}(i)$ (cf. [Cha06]). It starts with the highest colour which is $k := 2 \cdot (2 \cdot n + 3)$ in our case. The first observation is that the algorithm will direct Player 0 to reach C_k. This is only possible if the play in ρ_2 visits each of the vertices u_i not visited in ρ_1. Secondly, Player 0 will be directed in ρ_2 to skip all vertices u_i already visited in ρ_1. This is because the computed attractor strategy chooses the shortest way into C_k. The set of u_i-vertices visited in ρ_1 uniquely determines the strategy for Player 0 after the visit to x. For example, if Player 1 moves from v directly to x without visiting any u_i then Player 0 will play the strategy that visits each vertex u_i to reach C_k. This means we get the play $\rho = vxu_1v_1 \ldots u_nv_ny^\omega$. If Player 1 visits each u_i before moving to x then we get e.g. $\rho = vu_1v \ldots u_nvxv_1 \ldots v_ny^\omega$ because in this case the shortest way into C_k is to skip all vertices u_i. Altogether, for each i the vertex u_i is visited between x and y if and only if it has not been visited between v and x. Any of the 2^n subsets in U_v yields a different strategy each requiring its own state in the strategy automaton. Hence, this automaton will be of exponential size.

To see item 2. of the lemma we have to determine the classes of $\approx_S$. First we reduce the problem of computing $\approx$ for the game automaton $\mathcal{A}$ of Γ'_n to the problem of computing the standard equivalence relation $\approx_{\mathcal{A}'}$ for a DFA $\mathcal{A}'$ uniquely determined by $\mathcal{A}$ (cf. Sect. 4 and [Löd01]). This yields for two given states that $(s_1, q) \approx (s_2, q) \iff L(\mathcal{A}_{(s_1,q)}) = L(\mathcal{A}_{(s_2,q)})$ which allows us to identify equivalence of states with winning the same plays from the corresponding vertices in the game. The automaton $\mathcal{A}_{(s_1,q)}$ is the same as $\mathcal{A}$ except for the initial state set to (s_1, q). From $\approx$ we compute $\approx_S$ and obtain that the set $S/_{\approx_S}$ has five elements and that all sets in U_v are equivalent. Table 1 gives a review of the five equivalence classes where the sets from U_v are all in class three.

Table 1. *Equivalence classes of* $\approx_S$

Class	Description		Representative
	Winning Plays	Losing Plays	
1	y^ω	None	Q
2	Reach y	otherwise	$\{x\}$
3	Stay in U_v or reach both x and y	otherwise	$\{v, u_1\}$
4	Reach both x and y	otherwise	$\{v, u_1, v_1, u_2\}$
5	None	y^ω	$\{y\}$

□

The above lemma shows that there exist infinite games where our algorithm yields a substantial reduction of the needed memory. In [Hol07] we also present a family of games where the situation is converse: After applying our algorithm the set of memory contents is still exponentially large. Nevertheless, we obtain a positional winning strategy for Player 0 in the original game by minimising the strategy automaton.

Generally speaking, when solving an infinite game via game reduction it is possible to use Alg. 7 for reducing the memory. The only requirement is the existence of a compatible, language-preserving equivalence relation to reduce the game automaton of Γ'. Additionally, the classical approach of minimising the strategy automaton can be executed after solving the simplified game.

4 Implementation

GASt is a tool for synthesis problems in infinite games (cf. [Wal03]). We have integrated our memory reduction algorithm for two types of winning conditions, Request-Response and Staiger-Wagner. Request-Response games are reducible to Büchi games. For the reduction of Büchi game automata we use the notion of delayed simulation presented in [EWS05]. It can be computed in time $\mathcal{O}(m \cdot \log(n))$.[1] This reveals a technique for memory reduction for any type of game which is reducible to a Büchi game, e.g. generalised Büchi or upwards-closed Muller. Staiger-Wagner games can be reduced to weak parity games. The corresponding weak parity game automaton is equivalent to a deterministic weak Büchi automaton (DWA). The minimisation problem for DWA can be solved in time $\mathcal{O}(n \cdot \log(n))$ by a reduction to the minimisation problem for standard DFA (cf. [Löd01]). Both algorithms and any further details on how to compute the used equivalence relations can be found in [Hol07]. Whereas GASt provides

[1] Let n denote the number of states and m the number of transitions.

algorithms based on both the enumerative and symbolic representation of the state space we have considered only the enumerative case. Due to the state space explosion the running time of our algorithm grows very rapidly. Sometimes considering only the reachable vertices of G' already yields a substantial reduction of the needed memory. In this case we have to restrict the definition of $\approx_S$ further. More on this can be found in Sect. 6.1 in [Hol07]. The following table summarises some computation results for the game from La. 8 for the values $n = 2, 3, 4$ with and without the memory reduction algorithm.

Table 2. *Computation results*

n	Q_n	Γ'_n				Memory Reduction		Γ''_n		
		Create	S_n	Q'_n	Solve	$\approx_S$	Quotient	S'_n	Q''_n	Solve
2	7	40 ms	69	699	68 ms	1.8 s	0.3 s	5	23	$\leq$ 5 ms each
3	9	39 ms	203	2906	0.31 s	31.5 s	3.7 s	5	29	
4	11	0.26 s	609	11291	3.15 s	684 s	56 s	5	35	

5 Conclusion

We have presented an algorithm that reduces the memory for implementing winning strategies in infinite games. It is based on the notion of game reduction. The idea is to compute an equivalence relation on the set S of memory contents where two memory contents are considered equivalent if, from them, Player 0 wins exactly the same plays. The actual reduction is carried out via a transformation to ω-automata. Our algorithm has as parameters a game reduction and a compatible equivalence relation which mainly determine the running time. In La. 8 we have given a family of Staiger-Wagner games for which Alg. 7 reduces the needed memory to constant size whereas (without this reduction) the minimisation algorithm for strategy automata returns a memory of exponential size. This is because the algorithm from [Cha06] computes a very complicated winning strategy for this family of examples. Our algorithm can also be applied to Muller and Streett games which can both be reduced to (strong) parity games. For the reduction of parity game automata we use the right-hand delayed simulation for alternating parity automata introduced in [FW06], whose computation, in our case, amounts to solving a Büchi game of size $\mathcal{O}(n^2 \cdot k)$ where n is the number of states of the game automaton and k the number of colours.

References

[BL69] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.

[Cha06] K. Chatterjee. Linear Time Algorithm for Weak Parity Games. Technical report, EECS Department, University of California, Berkeley, 2006.

[DJW97] S. Dziembowski, M. Jurdziński, and I. Walukiewicz. How Much Memory is Needed to Win Infinite Games? In *Proceedings of the 12th LICS*, pages 99–110, Washington - Brussels - Tokyo, 1997. IEEE.

[EWS05] K. Etessami, T. Wilke, and R. A. Schuller. Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. *SIAM Journal on Computing*, 34(5):1159–1175, 2005.

[FW06] C. Fritz and T. Wilke. Simulation Relations for Alternating Parity Automata and Parity Games. In *Proceedings of the 10th DLT*, volume 4036 of *LNCS*, pages 59–70. Springer, 2006.

[GH82] Y. Gurevich and L. Harrington. Trees, Automata and Games. In *Proceedings of the 14th STOC*, pages 60–65, San Francisco, CA., 1982.

[Hol07] M. Holtmann. Memory Reduction for Strategies in Infinite Games. Diploma Thesis (revised version), RWTH Aachen, 2007. http://www-i7.informatik.rwth-aachen.de/∼holtmann/.

[Hop71] J. E. Hopcroft. An $n \log n$-Algorithm for Minimizing States in a Finite Automaton. Technical report, Stanford University, Department of Computer Science, 1971.

[Hor05] F. Horn. Streett Games on Finite Graphs. *GDV*, 2005.

[Koh70] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, New York, 1970.

[Löd01] C. Löding. Efficient minimization of deterministic weak ω-automata. *IPL*, 79:105–109, 2001.

[Mul63] D. E. Muller. Infinite Sequences and finite machines. In *Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 3–16, Chicago, Illinois, 1963. IEEE.

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the 12th STACS*, volume 900 of *LNCS*, pages 1–13, Munich, Germany, 1995. Springer.

[Tho96] W. Thomas. Languages, Automata, and Logic. Technical report, Christian-Albrechts-Universität Kiel, Institut für Informatik und Praktische Mathematik, 1996.

[Tho02] W. Thomas. Infinite Games and Verification. In *Proceedings of the 14th CAV*, volume 2404 of *LNCS*, pages 58–64. Springer-Verlag, 2002.

[Wal03] N. Wallmeier. Symbolische Synthese zustandsbasierter reaktiver Programme. Diplomarbeit, RWTH Aachen, 2003.

[Wal04] I. Walukiewicz. A landscape with games in the background. In *Proceedings of the 19th LICS*, pages 356–366. IEEE Computer Society, 2004.

[WHT03] N. Wallmeier, P. Hütten, and W. Thomas. Symbolic Synthesis of Finite-State Controllers for Request-Response Specifications. In *Proceedings of the 8th CIAA*, volume 2759 of *LNCS*, pages 11–22. Springer, 2003.

[Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.