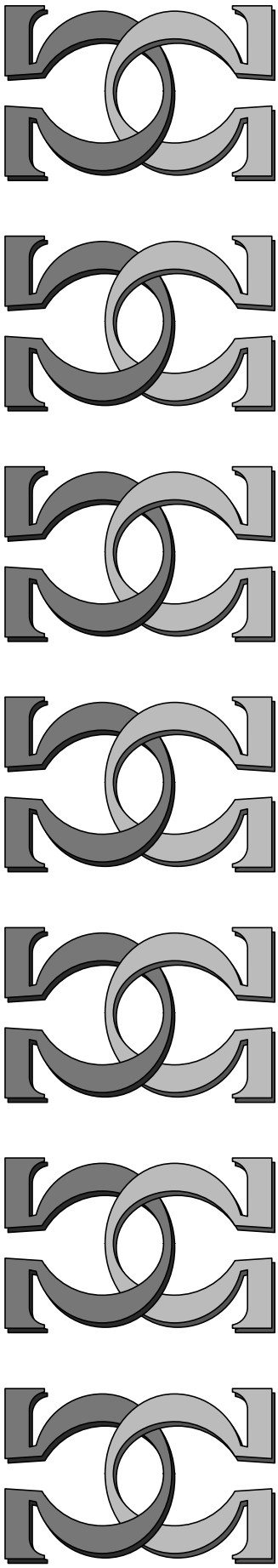

CDMTCS Research Report Series

Finite State Strategies in One Player McNaughton Games

Bakhadyr Khoussainov
Department of Computer Science
University of Auckland
Auckland, New Zealand

CDMTCS-189
May 2002

Centre for Discrete Mathematics and
Theoretical Computer Science

Finite State Strategies in One Player McNaughton Games

Bakhadyr Khoussainov
The University of Auckland, New Zealand

1 Introduction and Basic Concepts

In this paper we consider a class of infinite one player games played on finite graphs. Our main questions are the following: given a game, how efficient is it to find whether or not the player wins the game? If the player wins the game, then how much memory is needed to win the game? For a given number n, how does the underlying graph look like if the player has a winning strategy of memory size n?

The games we study can be seen as a restrictive case of McNaughton games first introduced in McNaughton's known paper [8]. In turn, McNaughton games belong to the class Büchi and Landweber games [1]. In these games one of the players always has a winning strategy by the result of Martin [7]. McNaughton in [8] proves that winners in his games have finite state winning strategies, that is strategies induced by finite automata, called LVR (last visitation record strategies) strategies; the name LVR is derived from LAR (last appearance record) first used in the known paper of Gurevich and Harrington [5].

McNaughton games can be used to develop game-theoretical approaches for many important concepts in computer science (e.g. models for concurrency, communication networks, update networks), to model computational problems (e.g. problems in known complexity classes such as P), and have close interactions with such fundamental notions as finite state machine or automata. For example, Nerode, Remmel and Yakhnis in a series of papers (e.g. [9] [10]) developed foundations of concurrent programming in which finite state strategies of McNaughton games are identified with distributed

concurrent programs. Another example is that McNaughton games may also be considered as models of reactive systems in which two players, Controller and Environment, interact with each other. Specifications of McNaughton games can then be associated with requirements put on the behaviour of the reactive systems. In this line, finite state winning strategies can be thought as programs satisfying the specifications.

In [3] Dinneen and Khoussainov use McNaughton games for modeling and studying structural and complexity-theoretical properties of update networks. Later, in [2] Bodlaender, Dinneen and Khoussainov generalize update networks by introducing the concept of relaxed update network and providing a p-time algorithm for detecting relaxed update networks. Ishihara and Khoussainov in [6] study another class of McNaughton games, called linear games, in which the winners can be detected efficiently with a given parameter.

Finally, we would like to mention the work of S. Dziembowski, M. Jurdzinski, and I. Walukievich [4]. They provide a detailed study of finite state winning strategies in McNaughton games, and show that the data structure associated with McNaughton's LVR strategies, in a certain natural sense, is an optimal one. For instance, they provide examples of McNaughton games Γ_n, $n \in \omega$, such that the size of the graph of Γ_n is $O(n)$, and every winning finite state strategy requires memory of size $n!$

In this paper we continue the line of work outlined above, and devote our study to a restrictive case of McNaughton games. We first begin with the following definition borrowed from [8]:

Definition 1 *A* **game** Γ *is a tuple* $(S \bigcup A, E, \Omega)$*, where:*

1. *The sets S and A are disjoint and finite, where S is the set of positions for Survivor and A is the set of positions for Adversary,*
2. *The set E of edges is such that $E \subseteq A \times S \bigcup S \times A$ and for all $s \in S$ and $a \in A$ there are $a' \in A$ and $s' \in S$ for which $(s, a'), (a, s') \in E$,*
3. *The set Ω is a subset of $2^{S \cup A}$.*

The graph (V, E), with $V = S \cup A$, is the **system** *or the* **graph of the game***, the set Ω is the* **specification***, and each set $U \in \Omega$ is a* **winning set***.*

In game Γ, a **play (from p_0)** is an infinite sequence $\pi = p_0, p_1, \ldots$ such that $(p_i, p_{i+1}) \in E$, $i \in \omega$. Survivor always moves from positions in S, while Adversary from A. Define $Inf(\pi) = \{p \mid \exists^{\omega} i(p = p_i)\}$. Survivor **wins** the play if $Inf(\pi) \in \Omega$; otherwise, Adversary wins. The **histories** are finite prefixes of plays. A **strategy** for a player is a rule that specifies the next move given a history of the play. Let f be a strategy for the player and p be a position. Consider all the plays from p which are played when the player follows the strategy f. We call these **plays consistent with f from p**.

Definition 2 *The strategy f for a player is a* **winning strategy** *from p if the player wins all plays from p consistent with f. In this case the player* **wins the game** *from p.* **To decide game** Γ *means to find the set of all positions, denoted by $Win(S)$, from which Survivor wins. The set $Win(A)$ is defined similarly*[1]*.*

McNaughton's algorithm in [8] that decides games is inefficient. In [9] Nerode, Remmel and Yakhnis improved the algorithm by deciding any given game Γ in $O(|V|!2^{|V|}|V||E|)$-time which is, of course, far from being efficient. A natural question arises as to find conditions, put either on the specifications or the systems, under which the games are decided efficiently. Here are related results.

A natural specification is to require Survivor to update every node of the system. Formally, Γ is an *update game* if $\Omega = \{V\}$. In [3] it is shown that update games can be decided in $O(|V||E|)$-time. Update games have been generalized in [2]. Namely, a game Γ is a *relaxed update game* if $U \bigcap W = \emptyset$ for all distinct $U, W \in \Omega$. It is proved that there exists an algorithm that decides relaxed update games in $O(|V|^2|E|)$-time. In [6] Ishihara and Khoussainov study *linear games* in which Ω forms a linear order with respect to the set-theoretic inclusion. They prove that linear games can be decided in polynomial time with parameter $|\Omega|$.

Clearly, in the results above, all the constraints are designed for specifications Ω but not for the structure (topology) of the system. In this paper we close this gap. We stipulate a constraint by not allowing Adversary to have any choices. In other words, we assume that from any given Adversary's node $a \in A$ Adversary can make at most one move. ¿From this point

[1] Any McNaughton game Γ is a Borel game. Hence, $Win(S) \bigcup Win(A) = S \bigcup A$.

of view, we study one player games. We investigate the complexity of finding the winners in these types of games, extracting winning strategies, and characterizing the topology of the game graphs in which Survivor wins.

Here is an outline of the paper. In the next section, Section 2, we introduce two basic concepts of this paper. One is that we formally define one player games. The other is the notion of a *finite state strategy*, - a strategy that can be induced by a finite automaton. We also define the concepts of global and local strategy. In Section 3, we provide an algorithm that decides any given one player game in linear time. In Section 4, we introduce games called basic games. Informally, these are building blocks of one player games. Given such a game, we provide efficient algorithms that construct finite automata inducing winning strategies. Finally, in the last section we provide a structural characterization of those basic games in which Survivor has a winning strategy induced by an automaton with a specified number of states. This characterization allows one to generate basic games that can be won by Survivor using strategies induced by n-state automata. We use notations and notions from finite automata, graphs, and complexity. All graphs are directed graphs. For a vertex (or equivalently node) v of a graph $\mathcal{G} = (V, E)$, we write $Out(v) = \{b \mid (v, b) \in\}$ and $In(v) = \{b \mid (b, v) \in E\}$. Cardinality of a set A is denoted by $|A|$.

2 One Player Games and Strategies

We concentrate on games where the topology of the graph games does not allow Adversary to make choices. We formalize this as follows. Let $\Gamma = (\mathcal{G}, \Omega)$ be a McNaughton game. Assume that $|Out(a)| = 1$ for each Adversary's node $a \in A$. This game is effectively a one player game, the player is Survivor, because Adversary has no effect on the outcome of any play. We single out these games in the following definition that ignores the set A of vertices, and assumes that all vertices are owned by one player, – Survivor.

Definition 3 *A* **one player game** *is a tuple $\Gamma = (V, E, \Omega)$, where V is the set of vertices, E is the set of directed edges such that for each $v \in$ there is v' for which $(v, v') \in E$, and $\Omega \subset 2^V$. The graph $\mathcal{G} = (V, E)$ is the* **system** *or the* **graph** *of the game, and Ω is the* **specification**.

Given a one player game Γ, there is only one player, – Survivor. A **run** (or **play**) from a given node v_0 of the system is a sequence $\pi = v_0, v_1, v_2, \ldots$

such that $(v_i, v_{i+1}) \in E$ for all $i \in \omega$. Survivor **wins** the play if $Inf(\pi) \in \Omega$. Otherwise, Survivor looses the play. Thus, each play is totally controlled by Survivor, and begins at node v_0. At stage $i+1$, given the history $v_0, \ldots, v_i$ of the play, Survivor makes the next move by choosing v_{i+1}. Survivor **wins the game from position** p if the player has a winning strategy that wins all the plays from p.

One can think of a one player game as follows. There is a finite state system represented as a graph whose nodes are the states of the system. The system fully controls all of its transitions. The system runs and tries to satisfy a certain task represented as a specification of the type Ω. In this metaphor, the system is Survivor and a winning strategy can be thought as a program satisfying the specification Ω. Deciding whether or not there is a winning strategy for Survivor can now be seen as a realizability of specification Ω for the given system.

Let $\Gamma = ((V, E), \Omega)$ be a one player game. We want to find finite state strategies that allow Survivor to win the game if this is possible. For this, we need to formally define finite state strategies. For game $\Gamma = (V, E, \Omega)$ consider an automaton $\mathcal{A} = (Q, q_0, \Delta, F)$, where V is the input alphabet, Q is the finite set of states, q_0 is the initial state, Δ maps $Q \times V$ to Q, and F maps $Q \times V$ into V such that $(v, f(q, v)) \in E$ for all $q \in Q$ and $v \in V$.

The automaton $\mathcal{A}$ **induces** the following strategy, called a **finite state strategy**. Given $v \in V$ and $s \in Q$, the strategy specifies Survivor's next move which is $F(s, v)$. Thus, given $v_0 \in V$, the strategy determines the run $\pi(v_0, \mathcal{A}) = v_0, v_1, v_2, \ldots$, where $v_i = F(q_{i-1}, v_{i-1})$ and $q_i = \Delta(v_{i-1}, q_{i-1})$ for each $i > 0$. If $Inf(\pi(v_0, \mathcal{A})) \in \Omega$, then $\mathcal{A}$ induces a winning strategy from v_0. When Survivor follows the finite state strategy induced by $\mathcal{A}$, we say that $\mathcal{A}$ **dictates** the moves of Survivor. To specify the number of states of $\mathcal{A}$ we give the following definition.

Definition 4 *A finite state strategy is an* n**-state strategy** *if it is induced by an* n *state automaton. We call* 1*-state strategies* **no-memory strategies**.

Let $\mathcal{A}$ be an automaton inducing a finite state winning strategy in game Γ. The strategy is a **global strategy** in the sense that $\mathcal{A}$ can process any given node of the system, and dictate Survivor's next move. This, generally speaking, means that $\mathcal{A}$ knows the global structure of the system, or at least in order to implement $\mathcal{A}$ one needs to know the structure of the game graph. Therefore implementing $\mathcal{A}$ could depend on processing the whole graph $\mathcal{G}$.

We now formalize the concept of local strategy in contrast to global strategies. An informal idea is the following. Given a game graph $\mathcal{G}$, assume that with each node $v \in V$ there is an associated automaton $\mathcal{A}_v$. The underlying idea is that $\mathcal{A}_v$ is a machine that knows v locally, e.g. all or some vertices adjacent to v and has no knowledge about the rest of the system.

Initially, all $\mathcal{A}_v$ are at their start states. The strategy for Survivor induced by this collection of machines is as follows. Say, Survivor arrives to node v. The player refers to machine $\mathcal{A}_v$ that based on its current state dictates Survivor's next move, changes its state, and waits until the player arrives to v the next time. The strategy induced in this way is a **local strategy**. Every local strategy is a finite state one. The basic idea is that implementing and controlling $\mathcal{A}_v$ at node v could be much easier and cheaper than implementing the whole machine that controls the system globally and dictates Survivor's moves all over the system.

3 Deciding One Player Games

This section provides a result that uses Tarjan's algorithm (see [11]) for detecting strongly connected graphs. Though the result is simple it shows a significant difference between times needed to decide McNaughton games in general case (see introduction) and in case of one player games. Here is our result:

Theorem 1 *There exists an algorithm that decides any given one player game* $\Gamma = (\mathcal{G}, \Omega)$ *in* $O(|\Omega| \cdot (|V| + |E|))$*-time.*

Proof. Let p be a node in V. We want to decide whether or not Survivor wins the game from p. Here is a description of our desired algorithm:

1. Check whether or not there is a $U \in \Omega$ such that for all $u \in U$ there exists a $u' \in U$ for which $(u, u') \in E$. Call such U **closed**. If there is no such U then declare that Survivor looses.

2. Check whether or not for every closed $U \in \Omega$ the subgraph $\mathcal{G}_U = (U, E_U)$, where $E_U = E \cap U^2$, determined by set U is strongly connected[2].

[2]A graph is **strongly connected** if there is path between any two nodes of the graph. Tarjan's algorithm detects whether or not the graph $\mathcal{G}_U$ is strongly connected in $O(|U| + |E_U|)$-time

If none of these graphs $\mathcal{G}_U$ is strongly connected, then declare that Survivor looses.

3. Let X be the union of all closed $U \in \Omega$ such that Γ_U is strongly connected. Check whether or not there is a path from p into X. If there is no path from p into X then declare that Survivor looses. Otherwise, declare that Survivor wins.

Note that it takes linear time to perform the first part of the algorithm. For the second part we use Tarjan's algorithm for detecting strongly connected graphs. Thus, for any closed $U \in \Omega$, apply Tarjan's algorithm to check whether or not $\mathcal{G}_U = (U, E_U)$ is strongly connected. The algorithm runs in $O(|U| + |E_U|)$-time. Hence overall running time for the second part is at most $c \cdot (|V| + |E|)$. For the third part, constructing X and checking if there is a path from p to X takes linear time. Thus, the algorithm runs at most in $O(c \cdot (|E| + |V|)$-time.

Now we need to show that the algorithm is correct. Assume that none of the winning conditions $U \in \Omega$ is closed. Consider any run $\pi = v_0, v_1, \ldots$ of the system. Clearly, there is no $U \in \Omega$ for which $Inf(\pi) = U$ because U is not closed. This shows that if there does not exist a closed $U \in \Omega$ then Survivor looses.

Now let U be a closed set. Consider the one player game $\Gamma_U = (\mathcal{G}_U, \{U\})$. Then it is not hard to see that Survivor wins this game if and only if the graph $\mathcal{G}_U = (U, E_U)$ is strongly connected. Now assume that for each closed U the graph $\mathcal{G}_U$ is not strongly connected. Consider any run $\pi = v_0, v_1, \ldots$ of the system and its infinity set $Inf(\pi)$. Clearly, if U is closed and $Inf(\pi) = U$ then this would mean that $\mathcal{G}_U$ is strongly connected. This contradicts the assumption.

Now assume that $X \neq \emptyset$ and there is a path from p into X. Then Survivor wins by using the following strategy. From node p reach X. Let q be the first position in X that has been reached. Then there is a strongly connected closed $U \in \Omega$ such that $q \in U$. Then Survivor wins by visiting each of the nodes in U infinitely often without ever leaving U. Assume that $X \neq \emptyset$ but there does not exist a path from p into X. Then by reasoning similar to above one sees that any run from p is not won by Survivor. Hence the algorithm is correct. □

4 Finite State Strategies for Basic Games

The proof of Theorem 1 shows that deciding one player games $\Gamma = (\mathcal{G}, \Omega)$ is essentially dependent on checking whether or not the graphs $\mathcal{G}_U = (U, E_U)$, where U is closed, are strongly connected. Therefore we single out the games that correspond to winning a single set $U \in \Omega$ in our next definition:

Definition 5 *A one player game $\Gamma = (\mathcal{G}, \Omega)$ is a* **basic game** *if $\Omega = \{V\}$.*

We begin with the next result showing that finding efficient winning strategies in basic games is computationally hard. By efficient winning strategy we mean an n-state winning strategy for which n is small.

Proposition 1 *For any basic game $\Gamma = (\mathcal{G}, \{V\})$, Survivor has a no-memory winning strategy if and only if the graph $\mathcal{G} = (V, E)$ has a Hamiltonian path. Therefore, finding whether or not Survivor has a no-memory winning strategy is NP-complete.*

Proof. Assume that the graph $\mathcal{G}$ has a Hamiltonian path $v_0, \ldots, v_n$. Then, it is clear that the mapping $v_i \to v_{i+1(mod(n+1))}$ establishes a no-memory winning strategy for Survivor. Assume now that in game Γ Survivor has a no-memory winning strategy f. Consider the play $\pi = p_0, p_1, p_2, \ldots$ consistent with f. Thus $f(p_i) = p_{i+1}$ for all i. Since f is a no-memory winning strategy we have $Inf(\pi) = V$. Let m be the least number for which $p_0 = p_m$. Then it is not hard to see that $V = \{p_0, \ldots, p_m\}$ as otherwise f would not be a winning strategy, and that the sequence $p_0, \ldots, p_m$ is a Hamiltonian path. □

The last two parts of the next theorem are of special interest. The second part of the theorem motivates the study of basic games in which Survivor can win with a fixed finite state strategy. This is done in the next section. The third part of the theorem provides a natural example of a local finite state strategy.

Theorem 2 *1. There exists an algorithm that decides any given basic game in $O(|E| + |V|)$-time.*

2. There exists an algorithm that for any given basic game in which Survivor is the winner provides an automaton with at most $|V|$ states that induces a winning strategy. Moreover, the algorithm runs in $O(|V|^2)$-time.

3. *There exists an algorithm that for any given basic game in which Survivor is the winner provides a finite state local winning strategy. Moreover, for each $v \in V$ the algorithm construct the automaton $\mathcal{A}_v$ in $O(|Out(v)|)$-time.*

Proof. Part 1 follows from Theorem 1. We prove Part 2. In the proof all additions are taken modulo n, where $n = |V|$. Let us list all nodes of the system $v_0, \ldots, v_{n-1}$. For each pair (v_i, v_{i+1}), $i = 0, \ldots, n-1$, introduce a state s_i, and find a path that $p_{i,0}, \ldots, p_{i,t_i}$ such that $p_{i,0} = v_i$ and $v_{i+1} = p_{i,t_i}$. The desired automaton in state s_i directs Survivor from v_i towards v_{i+1}, and as soon as v_{i+1} is reached the automaton changes its state from s_i to s_{i+1}. Finding a path from v_i to v_{i+1} takes linear time. Therefore constructing the desired automaton takes at most $O(|V|^2)$-time. This proves the second part of the theorem.

We now prove Part 3). Assume that Survivor wins Γ. We think of Γ as the graph $\mathcal{G}$ given in its adjacency list representation. Let v be a node of the system and $L(v) = \{p_0, \ldots, p_{r-1}\}$ be the list of all nodes adjacent to v. For node v we construct the automaton $\mathcal{A}_v$ with r states $s_0(v), \ldots, s_{r-1}(v)$, $s_0(v)$ being the initial state. The automaton acts as follows. When $\mathcal{A}_v$ is in state $s_i(v)$ and input is v the automaton dictates Survivor to move to p_{i+1} and changes its own state to $s_{i+1}(v)$, where addition is taken modulo r.

Let f be the strategy induced by the collection $\{\mathcal{A}_v\}_{v \in V}$ of automata. We need to prove that f is a winning strategy. Note that (V, E) must be strongly connected since Γ is won by Survivor by the assumption. Let π be the play consistent with f. Take $v \in Inf(\pi)$. Since $\mathcal{G}$ is strongly connected for every node $v' \in V$ there is a path $v_1, \ldots, v_n$ with $v_1 = v$ and $v_n = v'$. By the definition of f note that $v_1 \in Inf(\pi)$ because v occurs infinitely often. Reasoning in this way, we see that $v_2 \in Inf(\pi)$. By induction, we conclude that $v_n \in Inf(\pi)$. Therefore all the nodes in V appear in $Inf(\pi)$. The theorem is proved.

5 Finite State Strategies and Structure

In this section our goal consists of giving a characterization of the graph structure of the basic games $\Gamma = (\mathcal{G}, \{V\})$ in which Survivor has a winning n-state strategy. This characterization allows one to have a process that generates basic games won by Survivor using strategies induced by n-state

automata. Our main aim is to study the case when $n = 2$ as the case for $n > 2$ can be derived without much difficulty. So from now on we fix the basic game Γ, and refer to Γ as a game. The case when $n = 1$ is described in Proposition 1. The case when $n = 2$ involves a nontrivial reasoning.

Case $n = 2$. By the reasons that will be seen below (see the proof of Theorem 3) we are interested in graphs $\mathcal{G} = (V, E)$ such that $|In(q)| \leq 2$ and $|Out(q)| \leq 2$ for all $q \in V$. A path $p_1, \ldots, p_n$ in graph $\mathcal{G}$ is called a **2-state path** if $|In(p_1)| = |Out(p_n)| = 2$ and $|In(p_i)| = |Out(p_i)| = 1$ for all $i = 2, \ldots, n-1$. If a node q belongs to a 2-state path then we say that q is a **2-state node**. A node p is a **1-state node** if $|In(p)| = |Out(p)| = 1$ and the node is not a 2-state node. A path is a **1-state path** if each node in it is a 1-state node and no node in it is repeated.

We now define the operation which we call *Glue* operation that applied to finite graphs produces graphs. By a **cycle** we mean any graph isomorphic to $(\{c_1, \ldots, c_n\}, E)$, where $n > 1$ and $E = \{(c_1, c_2), \ldots, (c_{n-1}, c_n), (c_n, c_1)\}$. Assume that we are given a graph $\mathcal{G} = (V, E)$ and a cycle $\mathcal{C} = (C, E(C))$ so that $C \cap V = \emptyset$. Let P_1, ..., P_n and P'_1, ..., P'_n be paths in $\mathcal{G}$ and $\mathcal{C}$, respectively, that satisfy the following conditions: 1) These paths are pairwise disjoint; 2) Each path P_i is a 1-state path; 3) For each $i = 1, \ldots, n$, we have $|P_i| = |P'_i|$. The operation *Glue* has parameters $\mathcal{G}$, $\mathcal{C}$, P_1, ..., P_n, $\mathcal{P}'_1$, ..., P'_n defined above. Given these parameters the operation produces the graph $\mathcal{G}'(V', E')$ in which the paths P_i and P'_i are identified and the edges E and $E(C)$ are preserved. Thus, one can think of the resulted graph as one obtained from $\mathcal{G}$ and $\mathcal{C}$ so that the paths P_i and P'_i are glued by putting one onto the other. For example, say P_1 is the path p_1, p_2, p_3, and P'_1 is the path p'_1, p'_2, p'_3. When we apply the operation *Glue*, P_1 and P'_1 are identified. This means that each of the nodes p_i is identified with the node p'_i, and the edge relation is preserved. Thus, in the graph $\mathcal{G}'$ obtained we have the path $\{p_1, p'_1\}, \{p_2, p'_2\}, \{p_3, p'_3\}$. An important comment is the next claim whose proof can be seen from the definition:

Claim 1 *In the resulted graph $\mathcal{G}'$ each of the paths P_i is now a 2-state path.* □

The definition below defines the class of graphs that can be inductively constructed by means of the operation *Glue*.

Definition 6 *A graph $\mathcal{G} = (V, E)$ has a* **2-state decomposition** *if there is a sequence $(\mathcal{G}_1, \mathcal{C}_1)$, ..., $(\mathcal{G}_n\, C_n)$ such that $\mathcal{G}_1$ is a cycle, each $\mathcal{G}_{i+1}$ is*

obtained from the $\mathcal{G}_i$ and $\mathcal{C}_i$, and $\mathcal{G}$ is obtained from $\mathcal{G}_n$ and $\mathcal{C}_n$ by applying the operation Glue.

An example of a graph that admits a 2-state decomposition can be given by taking a union $\mathcal{C}_1,\ldots,\ \mathcal{C}_n$ of cycles so that the vertex set of each $\mathcal{C}_i$, $i = 1,\ldots,n-1$, has only one node in common with $\mathcal{C}_{i+1}$ and no nodes in common with other cycles in the list. We now need one additional definition:

Definition 7 *We say that the graph $\mathcal{G} = (V, E)$ is an* **edge expansion** *of another graph $\mathcal{G}' = (V', E')$ if $V = V'$ and $E' \subseteq E$.*

If Survivor wins a basic game $\Gamma = (\mathcal{G}, \{V\})$ with an n-state winning strategy then the same strategy wins the game $\Gamma' = (\mathcal{G}', \{V'\})$ where $\mathcal{G}'$ is an edge expansion of $\mathcal{G}$. Now our goal consists of proving the following theorem:

Theorem 3 *Survivor has a 2-state winning strategy in $\Gamma = (\mathcal{G}, \{V\})$ if and only if $\mathcal{G}$ is an edge expansion of a graph that admits a 2-state decomposition.*

Proof. Assume that Survivor has a 2-state winning strategy induced by the automaton $\mathcal{A} = (Q, \Delta, s_1, F)$. We need to produce a 2-state decomposition of a graph whose edge expansion is $\mathcal{G}$. Consider the play $\pi(p_1, \mathcal{A})$ dictated by $\mathcal{A}$. We can record the play, taking into account the sequence of states the automaton $\mathcal{A}$ goes through, as the following sequence $(p_1, s_1), (p_2, s_2), (p_3, s_3), \ldots,$ where s_1 is the initial state, $s_{i+1} = \Delta(s_i, p_i)$ and $p_{i+1} = F(s_i, p_i)$ for all $i \in \omega$. Note that the sequence $(p_1, s_1), (p_2, s_2), (p_3, s_3), \ldots$ is eventually periodic, that is, there are $i < j$ such that $(p_{i+n}, s_{i+n}) = (p_{j+n}, s_{j+n})$ for all $n \in \omega$. Therefore without loss of generality we may assume that the sequence

$$(p_1, s_1), (p_2, s_2), (p_3, s_3), \ldots, (p_k, s_k) \qquad (\star)$$

is, in fact, the period. Thus, $(p_{k+1}, s_{k+1}) = (p_1, s_1)$ and all pairs in $(\star)$ are pairwise distinct. Note that the set V of vertices coincides with $\{p_1, \ldots, p_k\}$ because $\mathcal{A}$ induces a winning strategy. Moreover, no p_i in the period $(\star)$ appears more than twice since $\mathcal{A}$ is a 2-state automaton.

An edge (p, q) of the system is an **$\mathcal{A}$-edge** if $p = p_i$ and $q = p_{i+1}$ for some i. Thus, an $\mathcal{A}$-edge is one used by $\mathcal{A}$ infinitely often. Consider the basic game $\Gamma(\mathcal{A})$ that occurs on the graph $(V, E(\mathcal{A}))$, where $E(\mathcal{A})$ is the set of all $\mathcal{A}$-edges. Clearly, $\mathcal{A}$ induces a winning strategy in game $\Gamma(\mathcal{A})$. Therefore, we always assume that $E = E(\mathcal{A})$.

If all p_is appearing in the period $(\star)$ are pairwise distinct then our theorem is true. Therefore we assume that there is at least one $p \in V$ that appears in $(\star)$ twice. Let $i < j$ be positions in the period $(\star)$ such that $p_i = p_j$ and no node of the system appears twice between these positions. Thus, we have the cycle $p_i, p_{i+1}, \ldots, p_j$ in the graph $\mathcal{G}$. We denote it by $\mathcal{C} = (C, C(E))$. Let now $p_n \in C$ be such that $(x, p_n) \in E$ and $x \notin C$. Hence, in the sequence $(\star)$ the automaton $\mathcal{A}$ dictates Survivor to move from x into p_n at some point of the play. Let p_m be be the first position at which Survivor chooses an edge not in $\mathcal{C}$ after moving from x into p_n. Thus, there is a y such that $(p_m, y) \in E \setminus E(C)$.

We claim that the path $p_n, \ldots, p_m$ is a 2-state path. Indeed, clearly $|In(p_n)| = |Out(p_m)| = 2$. This guarantees that each p in the path $p_n, \ldots, p_m$ appears twice in the period $(\star)$. Assume that $|In(p)| = 2$ for some p appearing strictly between p_n and p_m. This means there is a z such that $(z, p) \in E$, and this edge is an $\mathcal{A}$-edge. Therefore p must appear in $(\star)$ more than twice. We conclude that the assumption $|In(p)| = 2$ is incorrect. The case $|Out(p)| = 2$ can not happen because of the choice of p_m. Thus, $p_n, \ldots, p_m$ is a 2-state path.

Let P_1, …, P_t be all 2-state paths appearing in the path $p_i, \ldots, p_j$. These paths are pairwise disjoint. Consider the graph $\mathcal{G}(i,j) = (V(i,j), E(i,j))$, where $V(i,j)$ is obtained from V by removing all the vertices that belong to $C \setminus (P_1 \cup \ldots \cup P_n)$, and $E(i,j)$ is obtained by removing all the edges (a,b) from E if the edge (a,b) occurs in the path $p_i, p_{i+1}, \ldots, p_{j-1}, p_j$ and a and b belong to distinct paths P_1, …, P_t. Let $\Gamma(i,j)$ be the basic game played on the graph $(\mathcal{G}(i,j), \{V(i,j)\})$.

Lemma 1 *The graph $\mathcal{G}$ is obtained from the graph $\mathcal{G}(i,j)$, the cycle $\mathcal{C}$, and paths $P_1, \ldots, P_n$ by using the operator Glue. Moreover, Survivor has a 2-state winning strategy in the basic game $\Gamma(i,j)$.*

The first part of the lemma follows without difficulty. For the second part, we construct a 2-state automaton $\mathcal{A}'$ that induces a winning strategy in game $\Gamma(i,j)$. An informal description of $\mathcal{A}'$ is as follows. The automaton $\mathcal{A}'$ simulates $\mathcal{A}$ except the following case: Assume that the automaton $\mathcal{A}'$ is in state s and reads input $q \in P_i$ such that Survivor is dictated to stay in C but leave P_i. In this case $\mathcal{A}'$ behaves as $\mathcal{A}$ would behave when leaving C. More formally, assume that $\mathcal{F}(s,q) \in C \setminus P_i$ and $F(s',q) \notin C$ and (q,s') occurs in $(\star)$. In this case, $\Delta'(s,q) = \Delta'(s',q) = \Delta(s',q)$ and $F'(s,q) =$

$F'(s', q) = F(s', q)$. Now note that the play consistent with $\mathcal{A}'$ never reaches those nodes which are in $C \setminus (P_1 \cup \ldots \cup P_t)$ but assumes any other node infinitely often. The reason is that each node in $C \setminus (P_1 \cup \ldots \cup P_t)$ is a 1-state node, and appears in $(\star)$ once. The lemma is proved.

The size of $\mathcal{G}$ has now been reduced. Recursively, replace Γ with $\Gamma(i, j)$ and apply the same reasoning to Γ. Thus, $\mathcal{G}$ has a 2-state decomposition. We proved the theorem in one direction.

For the other direction, we need to show that if $\mathcal{G}$ in game $\Gamma = (\mathcal{G}, \{V\})$ is an edge expansion of a graph admitting a 2-state decomposition then Survivor has a 2-state winning strategy. Without loss of generality, we assume that $(\mathcal{G}_1, \mathcal{C}_1)$, ..., $(\mathcal{G}_n\, C_n)$ is a 2-state decomposition of $\mathcal{G}$. By induction on n we prove that Survivor wins game Γ.

For $n = 1$, $\mathcal{G}$ is simply a cycle. Hence Survivor wins $\mathcal{G}$ by a no-memory strategy. Assume now that Survivor has a 2-state winning strategy induced by an automaton $\mathcal{A}_n$ to win the game $\Gamma_n = (\mathcal{G}_n, \{V_n\})$, and $\mathcal{G}$ is obtained from $\mathcal{G}_n$ and the cycle $\mathcal{C}_n$ by using *Glue* operation on paths $P_1, \ldots, P_t$ and $P'_1, \ldots, P'_t$. Another inductive assumption is that $E(\mathcal{A}_n) = E_n$.

Lemma 2 *Consider the period $(\star)$ corresponding to the winning run $\pi(p, \mathcal{A}_n)$ in game $\mathcal{G}_n$. Then for each i, $1 \leq i \leq n$, each node in P_i appears in $(\star)$ just once.*

Indeed, take P_i and $x \in P_i$. Since P_i is a 1-state path either there is a path $p, \ldots, x$ such that $|Out(p)| = 2$ and $|Out(y)| = |In(y)| = 1$ for all nodes y between p and x, or there is a path $p, \ldots, x, \ldots, q$ for which $|In(p)| = 2$, $|In(q)| = 2$ and $|Out(y)| = |In(y)| = 1$ for all nodes y strictly between p and q.

In the first case, assume that x appears twice in $(\star)$. This mean p must appear in $(\star)$ twice, and Survivor must move along the path $p, \ldots, x$ at least twice. However, $|Out(p)| = 2$, and therefore Survivor at some position in the sequence $(\star)$ must choose the node adjacent to p but not in the path $p, \ldots, x$. Thus, p appears in the sequence $(\star)$ three times, which is a contradiction. In the second case, a contradiction is obtained in a similar manner by showing that q appears at least three times in the sequence $(\star)$.

The lemma above tells us that one can implement the automaton $\mathcal{A}_n$ in such a way that $\mathcal{A}_n$ does not change its states while running through each path P_i. Based on this assumption we now describe the automaton $\mathcal{A}_{n+1}$.

First, $\mathcal{A}_{n+1}$ runs through the cycle $\mathcal{C}$, then $\mathcal{A}$ simulates $\mathcal{A}_n$ by visiting all the nodes in $\mathcal{G}_n$ without entering nodes in $C \setminus (P_1 \cup \ldots \cup P_t)$. The theorem is proved. □

Now we outline how to generalize the case for $n > 2$. We are interested in graphs $\mathcal{G} = (V, E)$ such that $|In(q)| \leq n$ and $|Out(q)| \leq n$ for all $q \in V$. A path $p_1, \ldots, p_n$ in graph $\mathcal{G}$ is called an **n-state path** if $|In(p_1)| = |Out(p_n)| = n$ and $|In(p_i)| < n$ and $|Out(p_i)| < n$ for all $i = 2, \ldots, n-1$. If a node q belongs to an n-state path then we say that q is a **n-state node**. A node p is a **(n-1)-state node** if $|In(p)| < n$ and $|Out(p)| < n$ and the node is not an n-state node. A path is a **(n-1)-state path** if each node in it is a (n-1)-state node.

We now define the operation which we denote by $Glue_n$ that applied to finite graphs produces graphs. Assume that we are given a graph $\mathcal{G} = (V, E)$ and a cycle $\mathcal{C} = (C, E(C))$ so that $C \cap V = \emptyset$. Let $P_1, \ldots, P_t$ and $P'_1, \ldots, P'_t$ be paths in $\mathcal{G}$ and $\mathcal{C}$, respectively, that satisfy the following conditions: 1) These paths are pairwise disjoint; 2) Each path P_i is a (n-1)-state path; 3) For each $i = 1, \ldots, t$, we have $|P_i| = |P'_i|$. The operation $Glue_n$ has parameters $\mathcal{G}$, $\mathcal{C}$, P_1, …, P_t, $\mathcal{P}'_1$, …, P'_t defined above. Given these parameters the operation produces the graph $\mathcal{G}'(V', E')$ in which the paths P_i and P'_i are identified and the edges E and $E(C)$ are preserved.

Claim 2 *In the resulted graph $\mathcal{G}'$ each of the paths P_i is now a n-state path.* □

The definition below defines the class of graphs that can be inductively constructed by means of the operation $Glue_n$.

Definition 8 *A graph $\mathcal{G} = (V, E)$ has an **n-state decomposition** if there is a sequence $(\mathcal{G}_1, \mathcal{C}_1)$, …, $(\mathcal{G}_k, C_k)$ such that $\mathcal{G}_1$ is a cycle, each $\mathcal{G}_{i+1}$ is obtained from the $\mathcal{G}_i$ and $\mathcal{C}_i$, and $\mathcal{G}$ is obtained from $\mathcal{G}_k$ and $\mathcal{C}_k$ by applying the operation $Glue_n$.*

The next theorem gives a characterization of games at which Survivor has n-state winning strategy. The proof follows the lines of the previous theorem:

Theorem 4 *Survivor has a n-state winning strategy in $\Gamma = (\mathcal{G}, \{V\})$ if and only if $\mathcal{G}$ is an edge expansion of a graph that admits an n-state decomposition.* □

6 Conclusion

This paper deals with a structural constraint put on the topology of the systems in McNaughton games. It would be interesting to pinpoint those intrinsic structural properties of McNaughton games that make them hard to decide. It also seems that there is a trade-off between structural and specification constraints. Rigid structural constraints allow to weaken specification constraints without effecting efficiency of deciding games (witness Theorem 1), and vice versa. Also, we would not be surprised if the first part of Theorem 2 can be done more efficiently. For this, one probably needs to have a deeper analysis of BFS algorithms. It seems to be an interesting problem to understand when efficient finite state strategies (e.g. no-memory or 2-memory strategies) can be extracted by using efficient time and space resources. Finally, we mention the relationship between our games and temporal logic (see [12]) not discussed in this paper; essentially, the specifications of McNaughton games can be expressed in temporal logic. In general, we expect many more results in the study of McNaughton games from complexity, structure, and logic points of view.

References

[1] J. R. Büchi, L.H. Landweber. Solving Sequential Conditions by Finite State Strategies. *Trans. Amer. Math. Soc.* 138, p.295-311, 1969.

[2] H.L. Bodlaender, M.J. Dinneen and B. Khoussainov. On Game-Theoretic Models of Networks, in Algorithms and Computation (ISAAC 2001 proceedings), LNCS 2223, P. Eades and T. Takaoka (Eds.), p. 550-561, Springer-Verlag Berlin Heidelberg 2001.

[3] M. J. Dinneen and B. Khoussainov. Update networks and their routing strategies. In *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science, WG2000*, volume 1928 of *Lecture Notes on Computer Science*, pages 127–136. Springer-Verlag, June 2000.

[4] S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. How much memory is needed to win infinite games? In Proceedings, *Twelth Annual IEEE Symposium on Logic in Computer Science*, p. 99-110, Warsaw, Poland, 1997.

[5] Y. Gurevich and L. Harrington. Trees, Automata, and Games, STOCS, 1982, pages 60–65.

[6] H. Ishihara, B. Khoussainov. Complexity of Some Infinite Games Played on Finite Graphs, to appear in *Proceedings of the 28th International Workshop on Graph-Theoretic Methods in Computer Science*, WG 2002, Checz Republic.

[7] D. Martin. Borel Determinacy. Ann. Math. Vol 102, 363-375, 1975.

[8] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[9] A. Nerode, J. Remmel, and A. Yakhnis. McNaughton games and extracting strategies for concurrent programs. *Annals of Pure and Applied Logic*, 78:203–242, 1996.

[10] A. Nerode, A. Yakhnis, V. Yakhnis. Distributed concurrent programs as strategies in games. Logical methods (Ithaca, NY, 1992), p. 624–653, *Progr. Comput. Sci. Appl. Logic*, 12, Birkhauser Boston, Boston, MA, 1993.

[11] R.E. Tarjan. Depth first search and linear graph algorithms. SIAM J. Computing 1:2, p. 146-160, 1972.

[12] M. Vardi. An automata-theoretic approach to linear temporal logic. Proceedings of the VIII Banff Higher Order Workshop. Springer Workshops in Computing Series, Banff, 1994.