CDMTCS
Research
Report
Series

Games on Graphs: Automata, Structure, and Complexity

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

Tomasz Kowalski
Advanced Institue of Science and
Technology, Japan

CDMTCS-207
January 2002

Games on Graphs: Automata, Structure, and Complexity

Bakhadyr Khoussainov[1] and Tomasz Kowalski[2]

[1] Computer Science Department, The University of Auckland, New Zealand
`bmk@cs.auckland.ac.nz`
[2] Japan Advanced Institute of Science and Technology, Japan
(on leave from Department of Logic, Jagiellonian University, Poland)
`kowalski@jaist.ac.jp`

1 Introduction and Basic Concepts

McNaughton in his known paper [7], motivated by the work of Gurevich and Harrington [4], introduced a class of games played on finite graphs. In his paper McNaughton proves that winners in his games have winning strategies that can be implemented by finite state automata. McNaughton games have attracted attention of many experts in the area, partly because the games have close relationship with automata theory, the study of reactive systems, and logic (see, for instance, [12] and [11]). McNaughton games can also be used to develop game-theoretical approach for many important concepts in computer science such as models for concurrency, communication networks, and update networks, and provide natural examples of computational problems. For example, Nerode, Remmel and Yakhnis in a series of papers (e.g., [8], [9]) developed foundations of concurrent programming in which finite state strategies of McNaughton games are identified with distributed concurrent programs.

McNaughton games are natural descriptions of reactive systems in which the interaction between Controller (often referred to as Survivor) and Environment (Adversary) are modelled as certain two-player games. Winning conditions in these games can be thought of as specification requirements that Controller must satisfy. Winning strategies for Controller are thus identified with programs satisfying the specifications. Deciding whether or not Controller wins a given game can be seen as answering the question whether or not a given specification is realizable. If it is, then constructing the winning strategy amounts to synthesizing a correct controller program. Further, minimalization of the memory size of the winning strategy for Controller corresponds to the optimization problem of a correct controller. Again, we refer the reader to [11] for more details.

Suppose you come across a McNaughton game. You will probably expect that the particular structure of the underlying system and the specification of winning conditions influence in some way the running times of algorithms that decide the game. Such an expectation would be natural, for it is known that many algorithms for deciding McNaughton games are not efficient and do not explicitly exploit either the structure of the underlying graphs or the form of winning conditions. An exception is Zielonka [13] that shows that winners of the McNaughton games have finite state strategies that depend on nodes that are called *useful*.

The main purpose of this paper is to pursue this line of investigation a little further in a number of cases. In particular, we provide examples of classes of games for which the algorithms that decide these games explicitly use the nodes at which one of the players has more than one choice to make a move. We begin with the following definition extracted from [7]:

Definition 1. *A* game Γ*, played between two players called Survivor and Adversary, is a tuple* $(S \cup A, E, \Omega)$*, where:*

1. *The sets S and A are disjoint and finite, with S being the set of positions for Survivor and A the set of positions for Adversary,*
2. *The set E of edges is such that $E \subseteq (A \times S) \cup (S \times A)$ 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 of winning conditions is a subset of $2^{S \cup A}$.*

The graph $\mathcal{G} = (V, E)$, with $V = S \cup A$, is the system *or the* graph of the game*, the pair 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 $\mathit{Inf}(\pi) = \{p \mid \exists^{\omega} i : p = p_i\}$. Survivor *wins* the play π if $\mathit{Inf}(\pi) \in \Omega$; otherwise, Adversary wins π. We will refer to finite initial segments of plays as *histories*. 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*[3]*.*

From the definitions above it is clear that all graphs we consider are bipartite and directed. It is customary in the context of games to refer to members of V as nodes rather than as more graph-theoretical *vertices*. For a node v of a graph $\mathcal{G} = (V, E)$, we write $Out(v) = \{b \mid (v, b) \in\}$ and $In(v) = \{b \mid (b, v) \in E\}$. Usually nodes of set A are denoted by a, and of set S by s, possibly with indices.

As we have already said, McNaughton's algorithm in [7] that decides games is inefficient. In [8] 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, still far from being efficient. S. Dziembowski, M. Jurdzinski, and I. Walukiewicz in [2] investigated questions related to the size of memory needed for winning strategies.

[3] Any McNaughton game Γ is a Borel game. Hence, by the known result of Martin (see [6]), Γ is determined. Therefore $Win(S) \cup Win(A) = S \cup A$.

In particular, they prove that for each n there is a game Γ such that the size of V is $O(n)$ and the memory size for finite state winning strategies for these games are at least n factorial. A related question is under which conditions—imposed either on the specifications or on the systems—the games are decided efficiently and the memory size for winning finite strategies are sufficiently small. While the present paper has some bearing on the above question, it is also a continuation of a research trend, which we briefly summarise in the next paragraph.

Dinneen and Khoussainov have used McNaughton games for modelling and studying structural and complexity-theoretical properties of update networks (see [1]). A game Γ is an *update game* if $\Omega = \{V\}$. The system (V, E) is an *update network* if Survivor wins the update game. Speaking informally, Survivor is required to update (i.e., visit) every node of the system as many times as needed. In [1] it is shown that update games can be decided in $O(|V|(|V| + |E|))$-time. Update games have been generalized in [3] to games in which the specification Ω contains more than one set. Namely, a game Γ is a *relaxed update game* if $U \cap W = \emptyset$ for all distinct $U, W \in \Omega$. It is proved that there exists an algorithm that decides relaxed update games in $O(|V|^2(|V| + |E|)))$-time. In [5] 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 also be decided in polynomial time with parameter $|\Omega|$.

Clearly, in the results above, all the constraints are specification constraints. In other words, the games are described in terms of certain properties of specifications from Ω. In addition, the results as they stand—and most of their proofs—do not explicitly show the interplay between the structure of the underlying systems (V, E), the running times of algorithms that decide games, and the specifications in Ω. We try to bridge this gap by explicitly showing how running times of algorithms that decide certain classes of games depend upon the structure of the systems and specifications.

Here is a brief outline of the paper. In the next section, we introduce *no-choice* games and present a simple algorithm that decides them in time linear on the size of the game. We also provide a result that shows how the structure of the no-choice games is involved in finding finite state winning strategies with a given memory size. In Section 3 we revisit update games, and provide an algorithm that explicitly uses information about the number of nodes at which Adversary can make a choice, i.e., the members of A with at least 2 outgoing edges. In Section 4, we consider games in which specifications in Ω are closed under union. For such *union-closed* games we provide a decision algorithm whose running time depends explicitly on some structural information about the underlying systems of games. We note that the main result of this section can be obtained from the determinacy result of Zielonka [13]. However, our proof is direct and simple and does not need to employ the full strength of Zielonka's determinacy theorem. The final section discusses some issues for future work.

2 No-choice games

We start off with games where the structure of the system forces one of the players to always make a unique choice at any given node of the player. Without lost of generality we can assume that this player is Adversary. Formally:

Definition 3. *A* no-choice game *is a McNaughton game $\Gamma = (V, E, \Omega)$ such that for all $a \in A$, $s_1, s_2 \in S$ if $(a, s_1), (a, s_2) \in E$ then $s_1 = s_2$.*

No-choice games are one player games, with Survivor as the sole player, because Adversary has no effect on the outcome of any play. Below we provide a simple procedure deciding no-choice games by using Tarjan's algorithm that detects strongly connected directed graphs[4]. The algorithm is simple but shows a significant difference between times needed to decide McNaughton games in general case and in case of no-choice games. The following is a simple observation.

Lemma 1. *If X is a strongly connected component of $\mathcal{G}$ in a no-choice game $\Gamma = (V, E, \Omega)$ and $|X| > 1$, then $Out(a) \subset X$ for every $a \in A \cap X$.* □

Let $\Gamma = (S \cup A, E, \Omega)$ be a no-choice game. Call a winning set $U \in \Omega$ *S-closed* if $Out(a) \subseteq U$ for every $a \in A \cap U$, and $Out(s) \cap U \neq \emptyset$ for every $s \in S \cap U$. Clearly, if π is a play won by Survivor in game Γ then $Inf(\pi)$ must be S-closed. Thus, the following lemma holds true:

Lemma 2. *Survivor wins the no-choice game Γ if and only if Survivor wins the game Γ' which arises from Γ by removing all not S-closed winning sets.* □

Let $U \in \Omega$ be an S-closed winning set. Consider the game $\Gamma(U)$ whose graph is the restriction to U of the graph of Γ, and whose set of winning conditions $\Omega(U)$ is $\{U\}$. Define the graph $\mathcal{G}(U) = (V(U), E(U))$, where $V(U) = S \cap U$, and $(x, y) \in E(U)$ if and only if $x, y \in V(U)$ and $(x, a), (a, y) \in E$ for some $a \in U \cap A$. Thus, in graph $\mathcal{G}(U)$ there is a path between nodes p and q if and only if there is a finite play $s_1, a_1, \ldots, a_{n-1}, s_n$ in $\Gamma(U)$ such that $p = s_1$ and $q = s_n$. The following is easy:

Lemma 3. *Survivor wins $\Gamma(U)$ iff the graph $\mathcal{G}(U)$ is strongly connected.* □

Now we are ready to prove the following theorem:

Theorem 1. *There exists an algorithm that decides any given no-choice game $\Gamma = (V, E, \Omega)$ in $O(|\Omega| \cdot (|V| + |E|))$-time.*

Proof. Let p be a node in V. Here is a description of a desired algorithm:

1. If there is no S-closed $U \in \Omega$ then declare that Survivor loses.

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

2. If none of these graphs $\mathcal{G}(U)$ for S-closed $U \in \Omega$ is strongly connected, then declare that Survivor loses.
3. Let X be the union of all S-closed $U \in \Omega$ such that the graph $\mathcal{G}(U)$ 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 loses. Otherwise, declare that Survivor wins.

It takes linear time to perform the first part of the algorithm. For the second part, use Tarjan's algorithm for detecting strongly connected graphs. Namely, for each S-closed set U apply Tarjan's algorithm to check if $\mathcal{G}(U)$ is strongly connected. Hence the overall running time for the second part is proportional to $|\Omega| \cdot (|V| + |E|)$. 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 $O(|\Omega| \cdot (|E| + |V|)$-time. The correctness of the algorithm is clear. $\square$

Thus, the proof of Theorem 1 shows that deciding no-choice games is essentially dependent on checking whether or not the graphs $\mathcal{G}(U) = (V(U), E(U))$, where U is S-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 4. *A* **basic game** Γ *consists of a directed graph* $\mathcal{G}$ *and player Survivor, where Survivor is the sole player.*

Given a basic game Γ, a **play** from a given node v_0 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 $Inf(\pi) = V$. Otherwise, Survivor looses the play. Thus, *Survivor* wins the basic game Γ iff the graph $\mathcal{G}$ is strongly connected.

Let Γ be a basic game. Our goal is to find finite state strategies that allow Survivor to win the game. For this, we need to formally define finite state strategies. Consider an automaton $\mathbf{A} = (Q, q_0, \Delta, F)$, 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 $\mathbf{A}$ **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 $\pi(v_0, \mathbf{A}) = v_0, v_1, v_2, \ldots$, 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 $Inf(\pi(v_0, \mathbf{A})) = V$, then $\mathbf{A}$ induces a winning strategy from v_0. When Survivor follows the finite state strategy induced by *Adversary*, we say that $\mathbf{A}$ **dictates** the moves of Survivor. To specify the number of states of $\mathbf{A}$ we give the following definition.

Definition 5. *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**.

The next result shows 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 Γ, Survivor has a no-memory winning strategy if and only if the graph $\mathcal{G} = (V, E)$ has a Hamiltonian cycle. 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 cycle $v_0, \ldots, v_n$. Then the mapping $v_i \rightarrow v_{i+1(mod(n+1))}$ 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 $V = \{p_0, \ldots, p_m\}$ as otherwise f would not be a winning strategy, and hence the sequence $p_0, \ldots, p_m$ is a Hamiltonian cycle. $\square$

It is not hard to see that there exists an algorithm running in $O(|V|^2)$-time 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 (just check if for all $x, y \in V$ there are paths connecting x to y and construct a desired automaton) . Therefore, the following seem natural: If a player wins the game at all, then how much memory is needed to win the game? For a given number n, what does the underlying graph look like if the player has a winning strategy of memory size n? We will provide answers, however, we will not give full proofs.

Our goal is to analyze the case when $n = 2$, that is when Survivor has a 2-state winning strategy, as the case for $n > 2$ can then be derived without much difficulty. The case when $n = 1$ is described in Proposition 1. The case when $n = 2$ involves some nontrivial reasoning.

Case $n = 2$. 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) The 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, 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 $\{p_1, p_1'\}, \{p_2, p_2'\}, \{p_3, p_3'\}$. It is easily checked that in the resulting graph $\mathcal{G}'$ each of the paths P_i is now a 2-state path.

Definition 6. *A graph $\mathcal{G} = (V, E)$ has a* **2-state decomposition** *if there is a sequence $(\mathcal{G}_1, \mathcal{C}_1)$, ..., $(\mathcal{G}_n, \mathcal{C}_n)$ 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.

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 $E' \subseteq E$.*

The following theorem provides a structural characterization of those strongly connected graphs which Survivor can win with 2-state winning strategies.

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

3 Update Games Revisited

Recall that a game of type $\Gamma = (V, E, \{V\})$ is called an update game; and Γ is an update network if Survivor wins the game. In this section all the games considered are update games. Our goal here is twofold. On the one hand, we describe a decomposition theorem for update networks. For a full proof of this theorem we refer the reader to [1]. On the other hand, we provide a new algorithm for deciding update networks so that the algorithm runs in linear time on the size of the graph given a certain set of of Adversary nodes as a parameter. More formally, let $\Gamma = (V, E, \{V\})$ be an update game. Let C be the set of all Adversary's nodes a such that $|Out(a)| > 1$. In other words, C contains all nodes at which Adversary has a choice of at least two different moves. We provide an algorithm deciding update games, so devised that its running time shows what role the cardinality of C plays in the decision procedure. Namely, our algorithm depends on the parameter $|C|$ and runs in the time $k \cdot (|V| + |E|)$, where k depends on $|C|$ linearly.

Let $\Gamma = (V, E, \{V\})$ be an update game. For any $s \in S$ define $Forced(s) = \{a \in A_s \mid |Out(a)| = 1\}$. Thus, $Forced(s)$ is the set where Adversary is 'forced' to move to s. Note the following two facts. If $\Gamma = (V, E, \{V\})$ is an update network then for every $s \in S$ the set $Forced(s)$ is not empty. Moreover, if $|S| \geq 2$, then for every $s \in S$ there exists an $s' \neq s$ and $a \in Forced(s)$ such that $(s', a) \in E$. An important definition is now this:

Definition 8. *In a game Γ, a* **forced cycle** *is a cycle $(a_k, s_k, \ldots, a_2, s_2, a_1, s_1)$ such that $a_i \in Forced(s_i)$ and $s_i \in S$.*

Forced cycles have even length, and are fully controlled by Survivor. Using the facts above one now can show that any update network Γ with $|S| > 1$ has a forced cycle of length ≥ 4. The lemma below tells us that forced cycles can be used to reduce the size of the underlying graph and obtain an equivalent game.

Lemma 4. *Let Γ be an update game with a forced cycle C of length ≥ 4. We can construct a game Γ' with $|V'| < |V|$ such that Γ is an update network iff Γ' is one.*

Proof (sketch). We construct the graph (V', E') for Γ'. Consider $C = (a_k, s_k, \ldots, a_2, s_2, a_1, s_1)$. For new vertices s and a define $S' = (S \setminus \{s_1, \ldots, s_k\}) \cup \{s\}$ and $A' = (A \setminus \{a_1, \ldots, a_k\}) \cup \{a\}$. The set E' of edges consists of all the edges in E but not the edges in C, edges of the type (s, a') if $(s_i, a') \in E$, or (a', s) if $(a', s_j) \in E$, or (s', a) if $(s', a_k) \in E$ for some $s_i, s_j, a_k \in C$. We also put (a, s) and (s, a) into E'. Thus, the cycle C has been reduced. It is routine to show that Γ is an update network iff Γ' is one. The idea is that Survivor controls C fully.

The operation of producing Γ' from Γ and forced cycle C is called the **contraction operation**. In this case we say that Γ is an **admissible extension of** Γ'. Thus, for Γ' to have an addmisible extension Γ' must possess a forced cycle of length 2. Clearly, there are infinitely many admissible extensions of Γ'.

Definition 9. *An update game $\Gamma = (\mathcal{G}, \{V\})$ has a* **forced cycle decomposition** *if there exists a sequence $\Gamma_1, \ldots, \Gamma_n$ such that $|S_1| = 1$, $|Out(s_1)| = |A_1|$, where $S_1 = \{s_1\}$, and each Γ_{i+1} is an admissible extension of Γ_i, and $\Gamma_n = \Gamma$. The sequence $\Gamma_1, \ldots, \Gamma_n$ is called a* **witness** *for the decomposition.*

Using the lemma and the definition above one can prove the following theorem. The theorem gives us a complexity result one the one hand, and a description of update networks on the other (see [1]).

Theorem 3. *There exists an algorithm that given a game Γ decides in $O(|V||E|)$ time whether or not the game is an update network. Moreover, an update game Γ is an update network if and only if it has a forced cycle decomposition.*

Now we show how the set $C = \{a \in A \mid |Out(a)| > 1\}$ can be used to decide update games. Our algorithm shows that if the cardinality of C is fixed then update games can be decided in linear time.

Let X be a subset of V in a game $\Gamma = (\mathcal{G}, \Omega)$. The graph $\mathcal{G}_X$ is defined as the subgraph of $\mathcal{G}$ whose vertex set is $V \setminus X$. We begin with the following simple lemma.

Lemma 5. *Assume that C is a singleton and $C = \{a\}$. If Survivor wins Γ then $In(a) \neq \emptyset$ and $Out(a)$ is contained in a strongly connected component of $\mathcal{G}_{\{a\}}$.*

Proof. It is clear that $In(a) \neq \emptyset$ as otherwise a would not be visited infinitely often in each play. Assume now that no strongly connected component of $\mathcal{G}_{\{a\}}$ contains $Out(a)$. There are x, y in $Out(a)$ such that the graph $\mathcal{G}_{\{a\}}$ does not contain a path from x into y. Consider the strategy that dictates Adversary to always move to x from the node a. Then, for any play π consistent with this strategy, $Inf(\pi)$ does not contain y. Hence, Survivor cannot win Γ.

We will generalize the lemma above to the case when the cardinality of C is greater than 1. In other words, Adversary has more than one node at which a choice can be made. Let $a_1, \ldots, a_n$ be all the nodes from C.

Lemma 6. *Assume that Survivor wins Γ. Then the following two properties hold true:*

1. *Each set $In(a_i)$ is not empty for $i = 1, \ldots, n$.*
2. *There is an element $b \in C$ such that $Out(b)$ is contained in a strongly connected component of $\mathcal{G}_C$.*

Proof. The first property is clearly true as otherwise Survivor could not win the update game Γ. We show how to prove the second property.

Take a_1. Assume that no strongly connected component of $\mathcal{G}_C$ contains $Out(a_1)$. Then there are x_1 and y_1 in $Out(a_1)$ such that $\mathcal{G}_C$ does not contain a path from x_1 to y_1. We make the following claims:

Claim 1. There is an $i > 1$ such that for every $z \in Out(a_i)$ there is a path from z to y_1 in the graph $\mathcal{G}_C$.

In order to prove the claim assume that for each a_i, $i > 1$, there is a z_i such that there is no path from z_i into y_1 in the graph $\mathcal{G}_C$. Define the following strategy for Adversary. Any time when a play comes to a_i, $i > 1$, move to z_i. At node a_1 move to x_1. It is not hard to see that in any play π consistent with this strategy the node y_1 does not belong to $Inf(\pi)$. This contradicts the fact that Survivor wins Γ. The claim is proved.

Without loss of generality we can assume that a_2 satisfies the condition of the claim above. If $Out(a_2)$ is contained in a strongly connected component then the lemma is proved. Otherwise, there are $x_2, y_2 \in Out(a_2)$ such that the graph $\mathcal{G}_C$ does not have a path from x_2 to y_2. We now prove the following.

Claim 2. There is an i with $1 \leq i \leq n$ such that for every $z \in Out(a_i)$ there is a path from z to y_2 in the graph $\mathcal{G}_C$. Moreover, for any such i it must be the case that $i > 2$.

Assume that a_1 satisfies the claim. Then in $\mathcal{G}_C$ there is a path from x_1 to y_2. Since a_2 satisfies Claim 1, in $\mathcal{G}_C$ there is a path from y_2 to y_1. Then, the path from x_1 through y_2 to y_1 is in $\mathcal{G}_C$ as well. This is excluded by our initial assumptions about a_1. Thus, $i \neq 1$. Certainly a_2 cannot satisfy Claim 2 either. Then, we complete the proof of Claim 2 by repeating the argument we employed to prove Claim 1.

Now, repeating inductively the above reasoning, and suitably renumbering nodes, we may assume that the sequence $a_1, \ldots, a_j$ has the following properties:

1. In each $Out(a_k)$, $k = 1, \ldots, j-1$, there are x_k, y_k such that the graph $\mathcal{G}_C$ contains no path from x_k to y_k.
2. For all $z \in Out(a_k)$ with $k = 2, \ldots, j$ there is a path from z to y_{k-1} in the graph g_C.
3. $\{a_1, \ldots, a_j\} \subseteq C$.

Now, if the set $Out(a_j)$ is not contained in a strongly connected component of $\mathcal{G}_C$ then there is an $a \in C$ such that for all $z \in Out(a)$ there is a path from z to y_j. Otherwise, one can again show that Adversary wins the game by never moving to y_j. Indeed, the assumptions above guarantee that all paths from x_j to y_j go through an Adversary's node. Therefore Adversary can avoid visiting the node y_j. This, however, contradicts the assumption that Survivor wins the game. Moreover, as above, it can be shown that $a \notin \{a_1, \ldots, a_j\}$. It follows that $j < n$. Thus, we can conclude that there is an $i \leq n$ such that $Out(a_i)$ is contained in a strongly connected component . The lemma is proved. $\square$

By virtue of the lemma above we can pick an $a \in C$ such that $Out(a)$ is contained in a strongly connected component of $\mathcal{G}_C$; denote the component by X_a. We construct a new update game $\Gamma' = (V', E', \{V'\})$ as follows:

1. $V' = (V \setminus X_a) \cup \{s\}$, where s is a new Survivor's node.
2. $E' = (E \cap V'^2) \cup \{(s, a) \mid \exists t \in X_a((t, a) \in E\} \cup \{(a, s) \mid \exists t \in X_a((a, t) \in E)\}$.

We refer to Γ' as the *reduced* game. The following lemma shows the use of reduced games.

Lemma 7. *Survivor wins the game Γ if and only if Survivor wins Γ'.*

Proof. Let f be Survivor's winning strategy in Γ. We describe Survivor's winning strategy f' in Γ' obtained by simulating f. When the play takes place outside $\{s\}$, then f' mimics the moves dictated by f for nodes outside X_a. When the play arrives at s then Survivor scans f forward up to the nearest point where f leaves X_a. Obviously such a point exists. Suppose f does so by requiring a move to a node $y \notin X_a$. Then in the game Γ' Survivor also moves to y. It is not hard to see that f' thus described is indeed a winning strategy.

Now assume that f' is Survivor's winning strategy in Γ'. We describe Survivor's winning strategy f in Γ by simulating f'. When the play takes place outside X_a then f mimics f'. When the play arrives at X_a, the strategy f tells Survivor to:

1. visit each node of X_a, then
2. find node y to which Survivor moves in game Γ' from node s according to strategy f', then
3. find $x \in X_a$ such that $(x, y) \in E$, and move to x inside X_a, then, finally,
4. from x move to y.

It is clear that f thus described is well-defined, i.e., Survivor can do what f requires. That f is indeed a winning strategy is not hard to see either. $\square$

Assume that an $a \in C$ is such that $Out(a)$ is contained in a strongly connected component X_a. Consider the reduced game Γ', and its underlying graph $\mathcal{G}' = (V', E')$. The natural mapping $h : V \to V'$ defined by putting $h(v) = s$ for all $v \in X_a$, and $h(v) = v$ otherwise, satisfies the following properties:

1. h is onto;
2. for all $x, y \in V$, $(x, y) \in E$ and $x, y \notin X_a$ implies that $(h(x), h(y)) \in E'$;
3. X is a strongly connected component of $\mathcal{G}_C$ if and only if $h(X)$ is strongly connected component of $\mathcal{G}'_C$.

These observations together with Lemma 6 yield that if Survivor wins Γ' then there is an $a \in C$ such that $Out(a)$ is contained in a strongly connected component of $\mathcal{G}'_C$. Moreover, by Lemma 7, we can reduce the sizes of strongly connected components to singletons one by one always arriving at an equivalent game. This amounts to a proof of the following lemma.

Lemma 8. *If Survivor wins the update game Γ then for any $a \in C$ the set $Out(a)$ is contained in a strongly connected component of $\mathcal{G}_C$.* □

Now we are ready to prove a theorem.

Theorem 4. *There exists an algorithm that, given an update game Γ with $|C| = n$, decides whether or not Γ is an update network in running time proportional to $n \cdot (|V| + |E|)$.*

Proof. Our procedure uses Tarjan's algorithm. We describe the basic steps of our procedure. Its correctness follows from previous lemmas.

1. If $C = \emptyset$ then apply Tarjan's algorithm to see if $\mathcal{G}$ is strongly connected. If $\mathcal{G}$ is strongly connected then Survivor wins; otherwise Adversary wins.
2. Find all strongly connected components, $X_1, \ldots, X_m$, of $\mathcal{G}_C$ by using Tarjan's algorithm.
3. If for some $a \in C$ the set $Out(a)$ is not contained in one of the strongly connected components $X_1, \ldots, X_m$, then Adversary wins.
4. Construct the graph $\mathcal{G}(C) = (V(C), E(C))$ as follows:
   (a) $V' = (V \setminus \bigcup_{a \in C} X_a) \cup \{s_1, \ldots, s_k\}$, where each s_i is a new Survivor's node, and $k = |C|$.
   (b) $E' = (E \cap V'^2) \cup \bigcup_{i=1}^{k} \{(s_i, a) \mid \exists t \in X_{a_i}((t, a_i) \in E\} \cup \bigcup_{i=1}^{k} \{(a, s_i) \mid \exists t \in X_{a_i}((a, t) \in E)\}$.
5. If Survivor wins the no-choice game $\Gamma(C) = (\mathcal{G}(C), \{V(C)\})$ then Survivor wins the game Γ. Otherwise, Adversary is the winner.

It is not hard to see that the algorithm runs in the time required. □

4 Union-Closed Games

In this section we focus on union-closed games, that is the games in which the specification set Ω is closed under the set-theoretic union operation. Structurally, it is a natural property, and we will use it in an essential way in the algorithm deciding these games.

Let Γ be a union-closed game. Consider $a \in A$ such that $|Out(a)| > 1$. Let S_0 and S_1 be pairwise disjoint nonempty sets that partition $Out(a)$. We define two games Γ_0 and Γ_1, where $V_i = V$, $\Omega_i = \Omega$, and $E_i = E \setminus \{(a,s) \mid s \in S_i\}$. In other words, in game Γ_i, moves of Adversary at node a are restricted to S_i. Here is the main theorem from which we will deduce an algorithm for deciding union-closed McNaughton games.

Theorem 5. *Let Γ be a union-closed game. Survivor wins the game Γ from p if and only if Survivor wins each of the games Γ_0 and Γ_1 from p.*

Proof. We need to prove the nontrivial direction. Let f_0 and f_1 be winning strategies of Survivor in games Γ_0 and Γ_1, respectively. We construct the following strategy f for Survivor in the original game Γ. Survivor that begins its play by first emulating f_0. Assume that $p, p_1, \ldots, p_n, a$ is the history of the play so far, and Survivor is emulating the strategy f_ϵ, where $\epsilon \in \{0,1\}$. Now consider Adversary's move from a. There are two cases.

Case 1. Adversary moves into S_ϵ. In this case, Survivor emulates f_ϵ until a is reached again.

Case 2. Adversary moves into $S_{1-\epsilon}$ by choosing an $s \in S_{1-\epsilon}$. In this case, Survivor scans the history $h = p, p_1, \ldots, p_n, a, s$ and "projects" it into the game $\Gamma_{1-\epsilon}$. The "projection" is obtained from h by forgetting all the detours that belong to the game Γ_ϵ. More formally, Survivor does the following:

- scans h from the beginning up to the first appearance of a followed by a $t \in S_\epsilon$;
- keeps scanning h up to the next appearance of a (there must be such, because h ends with a followed by $s \notin S_\epsilon$);
- forms h' by identifying the two appearances of a in the the sequence $a, s, \ldots, a$ and cutting off everything in between;
- repeats the procedure until the end of h.

The "projection" h' obtained this way will be a history of a play from $\Gamma_{1-\epsilon}$. The next move of Survivor then coincides with the move of Survivor in the game $\Gamma_{1-\epsilon}$ required by the the winning strategy $f_{1-\epsilon}$ for the next step after h'.

This strategy is a winning strategy. Indeed, consider a play π consistent with this strategy. If after a certain history h of π Adversary always moves to S_ϵ from a then the play π', obtained from π by removing the initial segment h, is a play in Γ_ϵ. Then, Survivor wins π by resorting to the strategy f_ϵ after h has been completed. By symmetry, Survivor also wins any play π in which Adversary almost always moves to $S_{1-\epsilon}$. Assume that Adversary switches infinitely often from S_0

to S_1 and back during the play. Then π can be written as $\pi = \alpha_1\beta_1\alpha_2\beta_2\ldots$, where $\pi_1 = \alpha_1\alpha_2\ldots$ is a play in Γ_0 consistent with f_0 and $\pi_2 = \beta_0\beta_1\ldots$ is a play in Γ_1 consistent with f_1. Therefore, $\mathit{Inf}(\pi) = \mathit{Inf}(\pi_1) \cup \mathit{Inf}(\pi_2)$. Since f_0 and f_1 are winning strategies for Survivor, we must have $\mathit{Inf}(\pi_1)$, $\mathit{Inf}(\pi_2) \in \Omega$. By union-closedness, we get $\mathit{Inf}(\pi) \in \Omega$. Thus, f is the winning strategy for Survivor as required. □

As a corollary we obtain a complexity-theoretic result for deciding union-closed games. To formulate it, we need yet another definition.

Definition 10. *Let $\Gamma = (V, E, \Omega)$ be a game. An* instance of Γ *is any game $\Gamma' = (V', E', \Omega')$ such that $V' = V$, $\Omega' = \Omega$, and $E' \subset E$ such that for every $a \in A$ the set $Out(a)$ with respect to E' has cardinality* 1.

Now we can state:

Theorem 6. *Let $\Gamma = (V, E, \Omega)$ be a union closed game. Let $a_1, \ldots, a_k$ be all nodes in A such that $n_i = |Out(a_i)| > 1$, $i = 1, \ldots, k$. Then the following is true:*

1. *Survivor wins Γ if and only if Survivor wins every instance of Γ.*
2. *Deciding the game Γ takes $O(n_1 \cdot \ldots \cdot n_k \cdot |\Omega| \cdot (|V| + |E|)$-time.*

Proof. Part 1 follows from Theorem 3. Part 2 follows from Theorem 1, the first part of the theorem, and the fact that there are exactly $n_1 \cdot \ldots \cdot n_k$ instances of Γ. □

Corollary 1. *If Survivor looses a union-closed game then Adversary has a no-memory winning strategy.*

Proof. By the theorem above, Adversary wins an instance of the game. Such an instance is itself a no-choice game in which Adversary wins, and the strategy naturally derived is a no memory strategy. □

We note that the corollary above can be obtained from the known determinacy result of Zielonka [13]. However, our proof is direct and simple and does not need to employ the full strength of Zielonka's determinacy theorem.

5 Concluding Remarks

In this paper we have shown that McNaughton games can be studied by exploiting the relationship between specifications and the structure of the underlying graphs. This seems to be a natural approach if one wants to find efficient algorithms for deciding different classes of McNaughton games and have practical implementations of winning finite state strategies. The ideas presented in this paper can clearly be generalized and produce new algorithms for deciding McNaughton games. For example, we plan to investigate the question how the cardinality of the set at which Adversary has more than one choice to make a move can affect the complexity of decision algorithms for McNaughton games.

References

1. 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.
2. S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. How Much Memory Is Needed to Win Infinite Games? in Proceedings of Twelfth Annual Symposium on Logic in Computer Science (LICS 97), p.99-118, 1997.
3. 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.
4. Y. Gurevich and L. Harrington. Trees, Automata, and Games, STOCS, 1982, pages 60–65.
5. H. Ishihara, B. Khoussainov. Complexity of Some Infinite Games Played on Finite Graphs, Proceedings of the 28th international workshop on graph-theoretic methods in computer science, WG 2002, Czech republic. to appear.
6. D. Martin. Borel Determinacy. Ann. Math. Vol 102, 363-375, 1975.
7. R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.
8. 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.
9. 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.
10. R.E. Tarjan. Depth first search and linear graph algorithms. SIAM J. Computing 1:2, p. 146-160, 1972.
11. W. Thomas. On the synthesis of strategies in infinite games. in: STACS 95 (E.W. Mayr, C. Puech, Eds.), Springer LNCS 900, 1-13, 1995.
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.
13. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200, 135-183, 1998.