Extracting Winning Strategies In Update Games

Imran Khaliq[1], Bakhadyr Khoussainov[1], and Jiamou Liu[2]

[1] Department of Computer Science, University of Auckland, New Zealand
[2] School of Computing and Mathematical Sciences
Auckland University of Technology, New Zealand
ikha020@aucklanduni.ac.nz, bmk@cs.auckland.ac.nz
jiamou.liu@aut.ac.nz

Abstract. This paper investigates algorithms for extracting winning strategies in two-player games played on finite graphs. We focus on a special class of games called update games. We present a procedure for extracting winning strategies in update games by constructing strategies explicitly. This is based on an algorithm that solves update games in quadratic time. We also show that solving update games with a bounded number of nonkdeterministic nodes takes linear time.
Keywords: Games, update networks, finite state strategies

1 Introduction

Games played on finite graphs have received increasing interest over the last few years. These games are natural models for reactive systems [7], concurrent and communication networks [10], and have close interactions with model checking, verification, automata and logic [5, 4, 6, 13]. Such a game is played between two players, Player 0 and Player 1, over a finite directed graph. The players play the game by moving a token along the edges of the graph. We assume that both players have perfect information, the game is turn-based and each move is deterministic.

There are two interrelated algorithmic problems in the investigation of games played on finite graphs. The first is concerned with finding algorithms that determine the winner of the game. The second is concerned with extracting a winning strategy for the winner. These two problems have been investigated for games with Muller winning conditions. We refer to these games as McNaughton games as McNaughton was the first who studied algorithmic properties of these games [8]. In his paper [8] McNaughton provided an algorithm that detects the winners of these games. Nerode, Remmel and Yakhnis analyzed and improved McNaughton's algorithm in [9]. A. Dawar and P. Hunter showed that detecting the winners in McNaughton games is PSPACE complete problem [11]. In addition, McNaughton in [8] proves that the winners possess winning strategies that can be simulated by finite state automata. The upper bound for the number of states of the automata is $n!$, where n is the number of nodes in the game graph. Dziembowski, Jurdzinski and Walukiewicz showed that $n!$ bound is sharp [3].

A natural question arises to single out special classes of McNaughton games for which there are efficient algorithms that (1) find the winners of the games, and (2) extract finite state winning strategies. In [2], Dinneen and Khoussainov investigated a subclass of McNaughton games called update games. They proved that the winner of a given update game can be found in quadratic time on the size of the input game. Various generalizations of update games have been studied in [1], all these generalizations are solved in polynomial time. We also point out that, using straightforward transformations, Büchi and reachability games can be turned into McNaughton games. It is a well known fact that these games can be solved in linear and quadratic time, respectively, and that the winners have memoryless winning strategies [5].

In this paper we investigate update games. In these games Player 0 aims to visit every node of the game graph infinitely often. Update games can be viewed as models of various types of networks in which a message needs to be passed among all members of the network. We pursue several goals: In Section 4 we provide a procedure that generates all the update games in which Player 0 is the winner (Theorem 2). This will be based on structural properties of the underlying game graphs. In Section 5, we present a procedure that, given an update game, explicitly constructs a winning strategy for the winner (Theorem 3). We will provide such an algorithm that runs in quadratic time in the worst case. The algorithm is based on the contraction operator first introduced in [2]. Finally, in the last section we describe an algorithm that detects the winner of update games in linear time given that the number of nondeterministic nodes of Player 1 is bounded by a constant (Theorem 4). We note this requires a new technique since the algorithm based on the contraction operator does not make use of non-deterministic nodes of Player 1.

2 Preliminaries

The games under study are of the form (V_0, V_1, E, Ω) where $(V_0 \cup V_1, E)$ is a finite directed bipartite graph, V_0 and V_1 partition the set $V = V_0 \cup V_1$, $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$, and $\Omega \subseteq 2^V$ is a set of *winning positions*. For each $v \in V$, let vE denote the set $\{v' \in V_0 \cup V_1 \mid (v, v') \in E\}$, the *successors* of v. For this paper we stipulate that $vE \neq \emptyset$ for all $v \in V$. For $\sigma \in \{0, 1\}$, we call nodes in V_σ *Player σ's nodes*.

Intuitively, the players play the game by moving a token along the edges of the graph. The token is initially placed on a node $v_0 \in V$. The play proceeds in rounds. At any round of the play, if the token is placed on a Player σ's node v, then Player σ chooses $u \in vE$, moves the token to u and the play continues on to the next round. Formally, a *play (starting from v_0)* is a sequence $\rho = v_0, v_1, v_2, \ldots$ such that $v_{i+1} \in v_i E$ for all $i \in \mathbb{N}$. We set $\mathsf{Inf}(\rho) = \{v \in V \mid \forall i \exists j > i : v_i = v\}$. Thus, any play is infinite and $\mathsf{Inf}(\rho) \neq \emptyset$. We say Player 0 *wins the play* ρ if $\mathsf{Inf}(\rho) \in \Omega$; otherwise, Player 1 wins the play. These games are called *McNaughton games* or games with Muller winning conditions. We simply refer to them as games.

A *strategy* for Player σ is a function that takes as input initial segments of plays $v_0, v_1, \ldots v_k$ where $v_k \in V_\sigma$ and outputs some v_{k+1} such that $v_{k+1} \in v_k E$. We will concentrate on finite state strategies, which is realized by a finite I/O (Mealy) automaton $\mathcal{S} = (Q, q_0, \delta)$ where Q is the finite set of states, $q_0 \in Q$ is the initial state, and $\delta : Q \times V_\sigma \to Q \times V_{1-\sigma}$ is the *transition function*. The strategy $\mathcal{S}$ is a *k-state strategy* if $|Q| = k$. One state strategies are also called *memoryless strategies*. Thus, a memoryless strategy for Player σ is simply a function $\mathcal{S} : V_\sigma \to V_{1-\sigma}$. A play $\rho = v_0, v_1, v_2, \ldots$ is *consistent* with $\mathcal{S}$ if there exists a sequence of states $q_0, q_1, q_2, \ldots$ such that for all $i \in \mathbb{N}$ we have the following: If $v_i \in V_\sigma$, then $\delta(q_i, v_i) = (q_{i+1}, v_{i+1})$; If $v_i \in V_{1-\sigma}$, then $q_{i+1} = q_i$. Thus, the strategy does not change its state when Player $(1-\sigma)$ makes moves. A strategy for Player σ is *winning* from node v_0 if assuming Player σ always acts according to the strategy, all plays starting from v_0 generated by the players are winning for Player σ. A game Γ is *determined* if one of the players has a winning strategy starting at any given node v of the game. *To solve a game* means to find all positions from which a given player wins.

3 Update games and their basic properties

A game (V_0, V_1, E, Ω) is an *update game* if $\Omega = \{V\}$. An *update network* is an update game where Player 0 wins the game from some node. we denote the update game by (V_0, V_1, E). Dinneen and Khoussainov in [2] discussed basic properties of these games. Our focus will be to give a refined analysis of the winning strategies in update games.

Let Γ be a game. The 0-*attractor* set of X, denoted $\mathsf{Attr}_0(X)$, is the set of all nodes $v \in V$ that satisfy the following: Player 0 has a memoryless strategy $\mathcal{S}$ such that every play consistent with $\mathcal{S}$ that begins from v eventually reach some node in X. It is well-known that the set $\mathsf{Attr}_0(X)$ for any $X \subseteq V$ can be computed in time $O(|V| + |E|)$ [9].

Proposition 1. *A game Γ is an update network if and only if $\mathsf{Attr}_0(\{v\}) = V$ for all $v \in V$.*

Proof. Let $V = \{v_1, v_2, \ldots, v_n\}$. If $\mathsf{Attr}_0(\{v_i\}) = V$ for all $v \in \{1, \ldots, n\}$, then Player 0 wins as follows: When v_i is visited, Player 0 applies the memoryless "attractor" strategy to visit v_{i+1} (here we let $n + 1 = 1$). □

It is clear that the winning strategy described in the above proof requires $|V|$ states. Proposition 1 gives us a simple algorithm that solves update games:

Corollary 1. *There exists an algorithm that solves update games $\Gamma = (V_0, V_1, E)$ in time $O(|V| \cdot (|V| + |E|))$. Furthermore, if Γ is an update network, then Player 0 wins with an $|V|$-state strategy. Otherwise, Player 1 wins with a memoryless strategy.* □

A *t-star network* is a game where $|V_0| = 1$, $|V_1 = t$ and the only node in V_0 is linked to every node in V_1 via an edge.

Lemma 1. *A game Γ where $|V_0| = 1$ is an update network if and only if it is a t-star network for some t.* □

Let Γ be an update game. By Corollary 1, if Γ is an update network then Player 0 has an n-state winning strategy. Our goal is to build more sophisticated finite state wining strategies for update games. To do this we recast some of the results from [2]. One of the important concepts in the analysis is the notion of a forced cycle defined below.

Definition 1. *Let Γ be an update game.*

1. *For a Player 0's node v define the set $\mathsf{Forced}(v) = \{u \mid v \in uE, |uE| = 1\}$. We say that Player 1 is* forced *to move from u to v if $u \in \mathsf{Forced}(v)$.*
2. *A* forced cycle *is a sequence of pairwise distinct nodes $x_1, y_1, x_2, y_2, \ldots, x_k, y_k$ such that $x_i \in V_0$, $y_i \in \mathsf{Forced}(x_{i+1})$ and $y_k \in \mathsf{Forced}(x_1)$ for all $i = 1, 2, \ldots, k$. A forced cycle of length 2 is called a* spike*. If x, y form a spike we denote it by $x \leftrightarrow y$.*

We would sometimes abuse the notation by referring to a forced cycle as a set of nodes. We now state several properties (without proofs) of an update network Γ with $|V_0| > 1$.

Lemma 2. *1. For all $v \in V_0$, $\mathsf{Forced}(v) \neq \emptyset$.*
2. *For every $v \in V_0$, there exists $w \in \mathsf{Forced}(v)$ and $u \in V_0$ such that $(u, w) \in E$ and $u \neq v$.*
3. *The game Γ contains a forced cycle which is not a spike.* □

Thus, every update game without forced cycles can not be an update network. Note that existence of forced cycles does not guarantee that the game is an update network.

4 Contraction and unfolding operators

Our goal is to introduce an operator that reduces the sizes of update games. This will be important for building finite state winning strategies in update games. Our definition follows [2].

Definition 2. *Let $\Gamma = (V_0 \cup V_1, E)$ be an update game and C be a forced cycle in Γ. The* contraction operator*, when applied to Γ and C, produces a new update game $\Gamma(C)$ as follows (See Fig. 1 for an example):*

1. *Contract all nodes in $C \cap V_0$ (resp. $C \cap V_1$) to a new node x(resp. y) of Player 0 (resp 1).*
2. *Put directed edges between x and y.*
3. *Replace all edges (u, v), where $u \in V \setminus C$ and $v \in C$, by the edges (u, x) if $v \in V_0$, otherwise replace the edge (u, v) with (u, y).*
4. *Replace all edges (u, v), where $u \in C$ and $v \in V \setminus C$, by the edge (x, v) (Note that $u \in V_0$ as C is a forced cycle).*

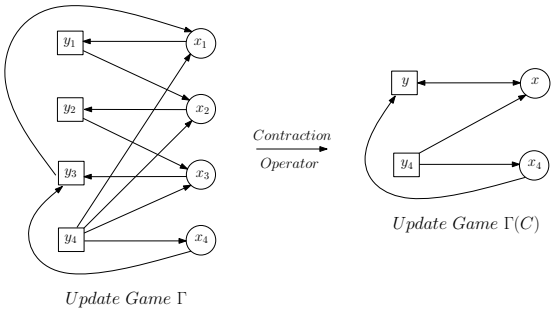

Fig. 1. In the update game Γ the forced cycle C is $\{x_1, y_1, x_2, y_2, x_3, y_3\}$. In the figure circles represent Player 0's nodes and squares represent Player 1's nodes.

5. *Keep all other nodes and edges in $V \setminus C$ intact.*

Lemma 3. *Let Γ be an update game with a forced cycle C. Then Γ is an update network if and only if $\Gamma(C)$ is an update network.*

Proof. Assume Player 0 wins the game $\Gamma(C)$ with winning strategy $\mathcal{S}$. In the following, we intuitively describe a winning strategy for Player 0 in Γ: Player 0 plays the game Γ by simulating a play π that is consistent with $\mathcal{S}$ in $\Gamma(C)$. If the play π visits nodes other than x and y, then Player 0 copies the movements of π in Γ. Once π moves from a node $v \notin \{x, y\}$ to x or y, then Player 0 in Γ moves to any node in $C \cap vE$. Then Player 0 first force the play to go around the cycle C and then when π leaves $\{x, y\}$ to some node u, Player 0 will move to a node $w \in V_0 \cap C$ such that $u \in wE$ and move to u. Since $\mathcal{S}$ is a winning strategy, it is easy to see that Player 0 also wins in the game Γ.

Conversely, assume Player 0 wins the game Γ with winning strategy $\mathcal{S}'$. Then Player 0 also wins $\Gamma(C)$ by simulating a play in Γ that is consistent with $\mathcal{S}'$. □

Suppose $\Gamma = (V_0, V_1, E)$ is an update game. By iteratively applying the contraction operator on forced cycles that are not spikes, we obtain a sequence of games $\Gamma_0, \Gamma_1, \ldots, \Gamma_k$ where $\Gamma_0 = \Gamma$, and $\Gamma_{i+1} = \Gamma_i(C_i)$ for all $i \in \{0, \ldots, k-1\}$, where C_i is a non-spike forced cycle in Γ_i. The last game Γ_k does not have a non-spike forced cycle. We call this sequence a *maximal contraction sequence of Γ*. Given Γ, finding a non-spike forced cycle C (or detecting such forced cycle does not exist), and constructing $\Gamma(C)$ take time $O(|V| + |E|)$. Therefore a maximal contraction sequence can be built in time $O(k \cdot (|V| + |E|))$. Combining Lemma 1, Lemma 2 and Lemma 3, we have:

Theorem 1 (Deciding Update Games). *There exists an algorithm that given an update game Γ constructs a maximal contraction sequence in time $O(k \cdot (|V| + |E|))$, where $k \leq |V|$. Moreover, Γ is an update network if and only if the last game in the sequence is a star network.* □

Our next goal is to extract a finite state winning strategies for update networks Γ using Theorem 1. For this we define an operator that, in some sense, inverses the contraction operator:

Definition 3. *Let Γ be an update game and $x \leftrightarrow y$ be a spike in Γ. Let C be any non-spike forced cycle that is disjoint from the graph of Γ. The unfolding operator, applied to $\Gamma, x \leftrightarrow y$ and C, proceeds in two steps as follows:*

1. **Replacement***: Replace the spike $x \leftrightarrow y$ with a new forced cycle C of length at least 4.*
2. **Enrichment***: First, associate edges of the forms (v, y) and (v, x) in Γ, where $v \notin \{x, y\}$, with sets $M_v^1 \subset C$ and $M_v^0 \subset C$ of Player 1 and Player 0 nodes, respectively. In addition, associate with each edge of the form (x, v), where $v \in V_1$, a set $M_v \subset C$ of Player 0 nodes. Now the enrichment operation proceeds as follows:*
   (a) Replace each edge (v, y) as above with the edges (v, u), where $u \in M_v^1$.
   (b) Replace each edge (v, x) as above with the edges (v, u), where $u \in M_v^0$.
   (c) Replace each edge (x, v) as above with the edges (u, v), where $u \in M_v$.

This resulting game is not uniquely determined and depends on the sets M_v^0, M_v^1, M_v. For notational convenience we suppress the parameters C, M_v^0, M_v^1 and M_v and denote the resulting game by $\Gamma(x \leftrightarrow y)$. We call it an unfolded game of Γ.

The proof of the following lemma follows from the definitions:

Lemma 4. *Let $x \leftrightarrow y$ be a spike in $\Gamma(C)$ obtained from contracting C. Then the following hold:*

1. *The unfolding operator applied to the update game $\Gamma(C)$, the spike $x \leftrightarrow y$, and forced cycle C produces the original game Γ, that is $\Gamma = \Gamma(C)(x \leftrightarrow y)$.*
2. *Γ is an update network if and only if $\Gamma(C)(x \leftrightarrow y)$ is an update network.* □

Theorem 1 and Lemma 4 allow us to construct update networks. Namely, start with a star network and consecutively apply the unfolding operation. All update games obtained in this way are update networks. Conversely, for every update network Γ there exists a sequence of unfolded games that starts from a star network such that the sequence produces Γ. We single out this observation as the following theorem:

Theorem 2 (Building Update Networks). *All update networks can be obtained by consecutively applying the unfolding operation to star-networks.* □

5 Extracting winning strategies

Let Γ be an update game and let $\Gamma_0, \ldots, \Gamma_k$ be a maximal contraction sequence of Γ. By Lemma 4, each Γ_{i+1} is an unfolding of Γ_i through a spike $x_i \leftrightarrow y_i$ in the game Γ_i. Assume that the game Γ_k is a t-star network. Our goal is to explicitly construct a finite state winning strategy for Player 0 by using this sequence.

Let $\mathcal{S} = (Q, q_0, \delta)$ be a finite state strategy in game Γ. We say that the spike $x \leftrightarrow y$ is *used by* $\mathcal{S}$ if for all state $q \in Q$ we have $\delta(q, x) = (q', y)$ for some $q' \in Q$. Otherwise, we say that the spike is *unused by* $\mathcal{S}$. For instance, when Γ is a t-star network, Player 0 has an obvious t-state winning strategy that visits all the Player 1 nodes in cyclic order. Every spike in Γ is used by the strategy. When the strategy $\mathcal{S}$ is clear, we drop the reference to $\mathcal{S}$ and simply say the spike is used or unused.

Lemma 5. *Assume that Player 0 has a t-state winning strategy $\mathcal{S}$ in the update game Γ. If the spike $x \leftrightarrow y$ is used by $\mathcal{S}$ then Player 0 has a t-state winning strategy in $\Gamma(x \leftrightarrow y)$. Otherwise Player 0 has a $(t+1)$-state winning strategy in $\Gamma(x \leftrightarrow y)$.*

Proof. We prove the lemma by explicitly constructing the strategy for Player 0 in $\Gamma(x \leftrightarrow y)$. The formal construction of the strategy is quite technical, thus we only provide a rough outline. Intuitively, Player 0 plays the game $\Gamma(x \leftrightarrow y)$ by simulating a play π in game Γ that is consistent with the strategy $\mathcal{S}$. Suppose the spike $x \leftrightarrow y$ is used by $\mathcal{S}$. When the play π visits x and $\mathcal{S}$ indicates Player 0 to visit y, Player 0 will visit a node $u_0 \in C$ and start to go around the cycle C while remaining in the same state. Upon returning to u_0, Player 0 changes its state and resume the simulation of π from the point where π has gone from x to y and back to x. This strategy of Player 0 still has t states as no new state is created for it to go around the cycle C. Suppose the spike $x \leftrightarrow y$ is not used by $\mathcal{S}$. Then the winning strategy of Player 0 in $\Gamma(x \leftrightarrow y)$ is defined similarly. However, we need an extra state so that once π visits x, Player 0 can "branch-off" the play in $\Gamma(x \leftrightarrow y)$ to go around the cycle C. Hence the strategy has $t+1$ states □

Consider the maximal contraction sequence $\Gamma_0 \ldots, \Gamma_k$ on Γ. Assume that the last game Γ_k is not a star-network. Our goal is to build a winning strategy for Player 1 in the original game. The strategy we construct will be memoryless as one would expect from Corollary 1.

Lemma 6. *Let Γ be an update game.*

1. *If there exists a node $v \in V_1$ without incoming edges then any memoryless strategy for Player 1 in Γ and in all unfolded games of Γ is a winning strategy.*
2. *Suppose that there exists a node $v \in V_0$ such that $\mathsf{Forced}(v) = \emptyset$. Player 1 has a memoryless winning strategy in Γ and all unfolded games of Γ.*
3. *Suppose that $|V_0| > 1$ and there exists a node $v \in V_0$ such that for all $w \in \mathsf{Forced}(v)$ no $u \neq v$ with $(u, w) \in E$ exists. Player 1 has a memoryless winning strategy in Γ and all unfolded games of Γ.* □

The two lemmas above and Theorem 1 give us the following theorem.

Theorem 3 (Extracting Winning Strategies). *Suppose that we are given a maximal contraction sequence $\Gamma_0, \ldots, \Gamma_k$ of an update game Γ.*

1. *Assume that Γ_k is a t-star network for some $t \in \mathbb{N}$. Player 0 has a $(t+m)$-state winning strategy in game Γ, where m is the total number of unused spikes that are unfolded in the sequence $\Gamma_0, \ldots, \Gamma_k$. Moreover, the strategy can be built in time proportional to m.*
2. *Assume that Γ_k is not a t-star network for any $t \in \mathbb{N}$. Player 1 has a memoryless winning strategy in game Γ. Moreover, the strategy can be built in time proportional to k.* □

Note that the winning strategies extracted by the procedure above depend on the number of contractions of forced cycles, and the number of states in this strategy may not be minimal. We remark that the problem of computing the minimal state winning strategies in update networks is NP-complete. For instance the Hamiltonian cycle problem can be reduced to finding a memoryless winning strategy problem in update networks. Indeed, given a directed graph G, subdivide every edge (u, v) into two edges (u, x) and (x, v) where x is a new node. This new graph is now the underlying graph for an update game $\Gamma(G)$. In $\Gamma(G)$, Player 0's nodes are the original nodes in G and Player 1's nodes are the new nodes. Clearly, the graph G has a Hamiltonian cycle if and only if Player 0 has a one-state winning strategy in the game $\Gamma(G)$.

6 Update games with a fixed number of nondeterministic nodes

A natural question arises if there exists a better algorithm that solves update games than the algorithm described in Theorem 1. For instance, one would like to know if it is possible to solve update games in linear time on the size of the graph. In a game Γ, we say a node u is *nondeterministic* if $|uE| > 1$. In this section we provide a linear-time algorithm to solve update games where there are at most k nondeterministic nodes of Player 1, where $k \geq 1$ is fixed. We denote the class of all such games by $\mathcal{U}_k$. Our algorithm takes a game Γ from the class $\mathcal{U}_k$, and reduces the game to an equivalent game from the class $\mathcal{U}_{k-1}$. The process may eventually produce a game from the class $\mathcal{U}_0$. The following is an obvious lemma that characterizes all the update networks from the class $\mathcal{U}_0$.

Lemma 7. *Let Γ be an update game from the class $\mathcal{U}_0$. Then Γ is an update network if and only if the underlying graph G of Γ is strongly connected.* □

Let Γ be an update game from the class $\mathcal{U}_k$, and let $b_0, b_1, \ldots, b_{k-1}$ be all nondeterministic nodes of Player 1 in Γ. Consider the graph H obtained by removing all these nondeterministic nodes from the underlying graph G. Let $C_0, \ldots, C_{t-1}$ be all the strongly connected components of H each of cardinality at least 2. We define a new update game Γ' called the *derivative* of Γ:

Definition 4. *For every strongly connected component C_i, collapse all Player 0 nodes in C_i into one node denoted by x_i, collapse all Player 1 nodes in C_i into one node denoted by y_i. Keep the other nodes of G intact. Note that some of the edges of G might collapse into one edge. The resulting graph is the underlying graph of the* derivative game *Γ'.(See Figure 2 for an example)*

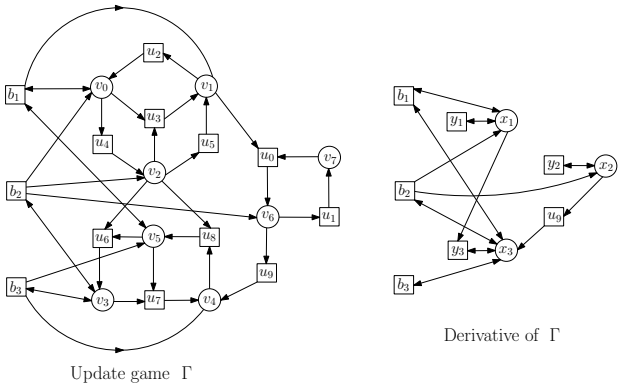

Fig. 2. Transformation of update game Γ to its derivative.

It is obvious that for each component C_i in the game Γ, Player 0 has a strategy f_i such that Player 0 using this strategy can stay in C_i forever and visit all the nodes of C_i infinitely often. We prove the next lemma using these strategies and a similar argument as the proof of Lemma 3.

Lemma 8. *The game Γ is an update network if and only if Γ' is an update network.* □

The next lemma gives us a sufficient condition for Player 1 to win the update game Γ.

Lemma 9. *If Γ is an update network then at least one of the nondeterministic nodes $b_0, \ldots, b_{k-1}$ becomes deterministic in the derivative game Γ'.*

Proof. Suppose no b_i where $i \in \{0, \ldots, k-1\}$ is deterministic in Γ'. We show that Player 1 has a memoryless winning strategy in Γ'. Consider the natural partial order $\leq$ defined on the spikes $x_1 \leftrightarrow y_1, \ldots, x_{t-1} \leftrightarrow y_{t-1}$ as follows. Say that $x_i \leftrightarrow y_i \leq x_j \leftrightarrow y_j$ if and only if there exists a path from C_j to C_i such that the path does not use the nondeterministic nodes $b_0, b_1, \ldots, b_{k-1}$.

We now define a strategy for Player 1 in Γ' as follows. Let $x \leftrightarrow y$ be a maximal spike in Γ'. Note that for any Player 1's node w with (w, x) there exists $w_x \neq x$ such that $(w, w_x) \in E'$. Define the strategy g_x so that $g_x(w) = w_x$. At all other nodes $u \neq w$, the strategy g_x is defined arbitrarily. This strategy is clearly a winning strategy for Player 1. □

Now consider the sequence of update games obtained by taking derivatives:

$$\Gamma_0 = \Gamma,\ \Gamma_1 = \Gamma_0',\ \Gamma_2 = \Gamma_1', \ldots$$

This process stops at stage s if either Γ_s has no nondeterministic nodes or Γ_s has the same number of nondeterministic nodes as Γ_{s-1}. We call the sequence $\Gamma_0, \Gamma_1, \ldots, \Gamma_s$ the *maximal derivative sequence* of Γ. From the three lemmas

above we have that Γ is an update network if and only if $\Gamma_s \in \mathcal{U}_0$ and the underlying graph G_s of Γ_s is strongly connected. Constructing Γ_{i+1} from Γ_i takes time proportional to $|V| + |E|$. This can be done using Tarjan's algorithm that computes strongly connected components[12]. Thus, we have the following theorem.

Theorem 4 (Deciding Update Games). *There exists an algorithm that given an update game Γ constructs the maximal derivative sequence in time $O(k \cdot (|V| + |E|))$, where k is the number of nondeterministic nodes of Player 1 in Γ. Moreover, the game Γ is an update network if and only if $\Gamma_s \in \mathcal{U}_0$ and G_s is a strongly connected component.* □

The following is an obvious corollary:

Corollary 2. *There is a linear time algorithm that solves update games from the class U_k.* □

References

1. Bodlaender, H.L., Dinneen, M.J., Khoussainov, B.: On Game-Theoretic Model of Networks. In: Proceedings of the 12th International Symposium on Algorithms and Computation. Eades, P., and Takaoka, T. (eds.) LNCS, vol. 2223, pp. 550-561. Springer, Heidelberg (2001)
2. Dinneen, M.J., Khoussainov, B.: Update Games and Update Networks. Journal of Discrete Algorithms. vol.1, issue 1. (2003)
3. Dziembowski, S., Jurdzinski, M., Walukiewicz, I.: How Much Memory is Needed to Win Infinite Games. In: *Proceedings of 12th Annual IEEE Symposium on Logic in Computer Science*. pp. 99-110 Warsaw Poland (1997)
4. Emerson, E.A., Jutla. C.S., Sistla, A.P.: On Model Checking for Fragments of μ-calculus. In: *Proceedings of the 5th International Conference on Computer Aided Verification*. LNCS 697, pp. 385-396 (1993)
5. Grädel, E., Thomas, W., Wilke, T.: Automata, logics, and infinite games: A Guide to Current Research. LNCS 2500, Springer, Heidelberg. (2002)
6. Ishihara, H., Khoussainov, B.: Complexity of Some Infinite Games Played on Finite Graphs. In: Proceedings of the 28th International Workshop on Graph-theoretic Methods in Computer Science. WG 2002
7. Mang, Y.C.: Games in open systems verification and synthesis. PhD Thesis, University of California at Berkeley (2002)
8. McNaughton, R.: Infinite Games Played on Finite Graphs. *Annals of Pure and Applied Logic*. 65 149–184 (1993)
9. Nerode, A., Remmel, J., Yakhnis., A.: McNaughton Games and Extracting Strategies for Concurrent Programs. *Annals of Pure and Applied Logic*. 78 203-242 (1996)
10. Nerode, A., Yakhnis, A., Yakhnis, V.: Distributed Concurrent Programs as Strategies in Games. *Logical Methods*. 624-653 (1992)
11. Hunter, P., Dawar, A.: Complexity Bounds for Regular Games. In: Proceedings of the 30th International Symposium on Mathematical Fondations of Computer Science (MFCS'05). LNCS 3618, Springer-Verlag. pp. 495–506 (2005)
12. Tarjan, R.: Depth-first Search and Linear Graph Algorithms. SIAM Journal on Computing. vol. 1, pp. 146–160 (1972)
13. Thomas, W.: Infinite games and verification. In: Proceedings of CAV 2002, LNCS 2404, pp. 58–64, Springer (2002)