How Much Memory is Needed to Win Infinite Games?[1,2]

Stefan Dziembowski[3]
std@mimuw.edu.pl

Marcin Jurdziński[3,4]
mju@mimuw.edu.pl

Igor Walukiewicz
igw@mimuw.edu.pl

Institute of Informatics[5]
Warsaw University

Abstract

We consider a class of infinite two-player games on finitely coloured graphs. Our main question is: given a winning condition, what is the inherent blow-up (additional memory) of the size of the I/O automata realizing winning strategies in games with this condition. This problem is relevant to synthesis of reactive programs and to the theory of automata on infinite objects. We provide matching upper and lower bounds for the size of memory needed by winning strategies in games with a fixed winning condition. We also show that in the general case the LAR (latest appearance record) data structure of Gurevich and Harrington is optimal. Then we propose a more succinct way of representing winning strategies by means of parallel compositions of transition systems. We study the question: which classes of winning conditions admit only polynomial-size blowup of strategies in this representation.

1 Introduction

We consider games played on (not necessarily finite) graphs coloured with a finite number of colours [15, 23]. The two players alternatively choose vertices forming an infinite path through the game graph. The winner is established by a Muller winning condition (defined on the finite set of colours).

These games can be seen as a special case of general Borel games and were first studied by Büchi and Landweber [4]. Gurevich and Harrington in their seminal paper [11] have shown that if a player has a strategy in such a game then there is a winning strategy for her that uses only finite information (*memory*) about the history of the play.

Since this result, infinite games became a central tool in the theory of automata on infinite objects [9, 13, 23, 19, 20]. They were also used in understanding logics of programs [9, 18, 21]. It was also observed that a game may be considered as a specification of a reactive system by viewing it as a description of all possible interactions between player 0 (Control) and player 1 (Environment). A winning condition in a game specifies some requirements on the behaviour of the reactive system (for example some liveness properties). In this metaphor, a winning strategy can be seen as a program satisfying the specification. Hence, in this framework, *strategy construction* amounts to *program synthesis* and deciding if there exists a winning strategy for player 0 is equivalent to the problem of *realizability of specifications* of reactive systems. These problems were studied for example in [1, 4, 14, 17].

In all these applications an important question is how much memory a strategy may need. As an example from automata theory, consider the complementation lemma for automata on infinite trees. There, in the construction of the automaton accepting the complement of the language, one takes as states subsets of the set of pairs: $\langle$state of the original automaton, memory state of the strategy$\rangle$. Hence the size of the required memory influences the size of the automaton for the complement. In particular it can change this size from single to double exponential. In the area of program logics, additional memory reflects in complications in proof constructions as for example in [21] or in tableau constructions [3, 6]. In these applications it is important to understand the structure of the additional memory. Until now essentially the only known memory structure were LAR's [11, 15, 23].

Most importantly, however, the question of memory size is crucial in the program synthesis. It measures the inherent blow-up of the size of a program satisfying a specification in terms of the size of the specification itself. In order

[1]A part of this research was carried out while the authors were visiting **B**asic **R**esearch in **C**omputer **S**cience, Centre of the Danish National Research Foundation.

[2]This work was partially supported by Polish KBN grant No. 8 T11C 002 11.

[3]Supported by **BRICS** Summer Student Programme.

[4]Supported by by Polish-Danish Student Exchange Programme.

[5]Address: Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland.

to understand feasibility of program synthesis, as described above, it is hence important to classify winning conditions according to the blow-up in the size of strategies they may incur.

In some applications it is necessary to deal with infinite game graphs, e.g., when proving complementation lemma for automata on infinite trees. When considering program synthesis, however, one usually takes into account game graphs of size at most polynomial in the number of colours in the winning condition. (Although finitely definable infinite size games also were considered in this context [5, 20, 22] and our results are also applicable there). As we will see the case when the size of the game graph is bounded in the number of colours is different and technically more difficult, hence it is appropriate to consider it separately. We will consider the problem: how much memory is needed to win games with a given class of winning conditions. We will consider both, the case when we bound the size of games by the number of colours in the condition and the case when we do not impose this bound.

It is well known that there exists a sequence of games $\langle \mathcal{G}_n \rangle_{n\in\mathbb{N}}$, each $\mathcal{G}_n$ of size $\mathcal{O}(n)$ and requiring strategy automata of size 2^n [20, 12]. Hence synthesized program may be exponentially larger than the specification. While it is still possible to imagine that we can deal with problems having worst case exponential time complexity, it is hard to imagine how we can cope with exponential space complexity. Indeed in the model checking community it was discovered that the space, not time, complexity is the major obstacle. To overcome the problem of big sizes of strategies we propose a new representation of strategies. This representation is inspired by a well known problem in model checking, namely, the state explosion problem.

Very often, reactive programs are presented as a parallel composition of sequential programs, where each sequential program is represented as a finite I/O automaton. It is well known that the size of the resulting space depends exponentially on the number of parallel components. This is a major obstacle in applying model checking techniques. Here, in the context of synthesis, we propose to use this phenomenon to our advantage. We suggest to describe, potentially big, strategy automaton as a parallel composition of, possibly small, components. We call this representation a *p-automaton*. There are examples when one can obtain exponential savings using p-automata. Indeed it is very difficult to come up with examples of games that need exponential size strategies in terms of p-automata. Such games must exist by a counting argument but giving a concrete example of such a game would imply a solution to one of the major problems about nonuniform complexity classes. The problem is to give an explicit description of a function $h : \{0,1\}^* \to \{0,1\}$ that is not computable in nonuniform NC^1.

1.1 Overview of the results

First we consider the case when we do not limit the size of game graphs in terms of the number of colours in the winning condition. We define the size of the memory of a strategy as number of states of an I/O automaton realizing the strategy. In this case we obtain matching upper and lower bounds for any fixed winning condition $\mathcal{F}$, i.e., we determine a number $m_{\mathcal{F}}$ such that:

1. For every game with the winning condition $\mathcal{F}$, memory of size $m_{\mathcal{F}}$ is sufficient for a winning strategy in this game.
2. There exists a game with the winning condition $\mathcal{F}$ for which every winning strategy requires memory of size at least $m_{\mathcal{F}}$.

If we restrict ourselves to game graphs of size polynomial in the number of colours in the winning condition then, clearly, the upper bound still holds. General lower bounds, however, do not follow from the above, because our games requiring memory of size $m_{\mathcal{F}}$ have graphs roughly of size $m_{\mathcal{F}}$ (which may be exponential in the number of colours). We, however, provide a particular lower bound, that establishes optimality of LAR's also in this setting:

3. There exists a sequence of games $\langle \mathcal{G}_n \rangle_{n\in\mathbb{N}}$, such that the game graph of $\mathcal{G}_n$ is of size $\mathcal{O}(n)$ and every winning strategy requires memory of size $n!$.

In the last section we consider the problem of representing strategies by means of p-automata. We show that the strategies in the games $\mathcal{G}_n$ from (3) can be represented by polynomial size p-automata. Indeed polynomial size p-automata realising winning strategies exist for all games with winning conditions that can be represented by a Turing machine working in a space polynomial in the number of colours used in the condition.

1.2 Comparison with the related work

Our upper bound theorem was inspired by the paper of Zielonka [23]. He mentions some savings in memory for winning strategies by considering his notion of "useful colours". This observation is sufficient for showing memoryless determinacy for games with Rabin winning conditions (see also [7, 13]). In general his bound is not optimal. Our upper bound is in some cases exponentially smaller than his. Zielonka's motivation comes from automata theory so he considers potentially infinite graphs. In this setting our upper and lower bounds for memory needed by winning strategies are optimal.

Memory requirements in the context of program synthesis were studied by Lescow [12]. He considers only game graphs of size linear in the number of colours and seeks a characterisation of winning conditions admitting polynomial-size strategies. Our upper bound subsumes all his examples of families of such winning conditions. Lescow provides also some $2^{\Omega(n)}$ lower bounds. We improve on them, showing an $\Omega(n!)$ lower bound. This closes the gap between previously known lower bounds and the factorial upper bound of LAR's. This may come as a slight surprise, because some claims to the contrary have been quoted (see [20], p. 9).

Finally let us mention the following algorithmic problem: given a game and a node, establish whether player 0 has a winning strategy from this node. McNaughton [15] gives an algorithm working in time exponential in the number of colours. In case the of games with winning conditions given in Rabin form, Emerson and Jutla [8] show that the problem is NP-complete. From our general upper bound on memory we infer a class of winning conditions for which the problem is in NP $\cap$ co-NP (see Corollary 12). This can be seen as a generalisation of the result due to Emerson, Jutla and Sistla [10], where NP $\cap$ co-NP upper bound was shown for games with Rabin chain (parity) winning condition. The latter problem is polynomially equivalent to the μ-calculus model checking [10]. Nerode, Remmel and Yakhnis [16] study classes of winning conditions for which there exist polynomial time algorithms of strategy synthesis. We provide (Corollary 13) a broader class of winning conditions allowing strategy synthesis in polynomial time.

2 Preliminaries

We consider infinite duration games played by two players (player 0 and player 1) on finitely coloured graphs called *arenas*. An *arena* is a tuple $\mathcal{A} = (V, V_0, V_1, E, C, \chi)$ where: (V, V_1, V_2, E) is a (not necessarily finite) bipartite directed graph with the partition V_0, V_1; C is a finite set of colours; and $\chi : V \rightharpoonup C$ is a partial *colouring function*. We assume that for every vertex of the arena there is at least one outgoing edge, i.e., for every $v \in V$ there is some $u \in V$, such that $(v, u) \in E$. Hence every path in the arena can be prolonged.

Remark: In order not to unnecessarily obscure our examples we will sometimes have arenas that are not bipartite graphs. It will be, however, always straightforward to restore the bipartite property by introducing some new vertices, but never more than linearly many in the size of the arena.

A *game* is a pair $\mathcal{G} = (\mathcal{A}, \mathcal{F})$, where $\mathcal{A}$ is an *arena*, and $\mathcal{F} \subseteq \mathcal{P}(C)$ is a *winning condition* for player 0 (in Muller form [19]). An *(infinite) play of the game* $\mathcal{G}$ is an (infinite) sequence $\pi = \langle v_0, v_1, v_2, \dots \rangle$ of nodes of $\mathcal{A}$, such that for every $i = 0, 1, 2, \dots$ we have $(v_i, v_{i+1}) \in E$.

Given an infinite play $\pi = \langle v_0, v_1, v_2, \dots \rangle$ of a game $\mathcal{G}$ we define the set of *frequent colours of* π (denoted by $\mathit{Inf}(\pi)$) as the set of colours that occur infinitely often in the sequence $\langle \chi(v_0), \chi(v_1), \chi(v_2), \dots \rangle$. An infinite play π is a *winning play* for player 0 if the set of its frequent colours is an element of the winning condition for player 0, i.e., $\mathit{Inf}(\pi) \in \mathcal{F}$.

A *strategy* for player 0 in a game $\mathcal{G}$ is a partial function $\zeta : V^* \rightharpoonup V_1$, defined for finite plays $\pi = \langle v_0, v_1, \dots, v_k \rangle$ with $v_k \in V_0$, such that $(v_k, \zeta(\pi)) \in E$. A play $\pi = \langle v_0, v_1, v_2, \dots \rangle$ from a node $v_0 \in V_0$ is *consistent* with the strategy if $\zeta(v_0, \dots, v_{2i}) = v_{2i+1}$ for $i = 0, 1, \dots$. A strategy ζ is *winning for player 0* from a vertex v_0 if every infinite play starting from v_0 consistent with ζ is winning for player 0.

A *winning set* for player 0 is the set of vertices of the arena from which there exists a winning strategy for player 0. It is easy to see that these strategies can be "merged" (see [15]), hence there is also one common winning strategy on the winning set.

Proviso: We will consider winning strategies for player 0. Thus w.l.o.g. we can assume that player 0 has a strategy to win from every vertex of the arena. To meet this requirement it is enough to restrict the arena to the winning set of player 0.

It turns out (see [11, 15, 23]) that for the games we consider the winning strategies can be realized by functions that do not have to refer to the whole *histories* of the play (i.e., the elements of the set V^*), but only to a finite information about the play so far. To make this precise we introduce the notion of *strategies with memory*.

Definition 1 (Strategy with memory) Let $\mathcal{A} = (V, V_0, V_1, E, C, \chi)$ be an arena. A *strategy automaton* is an I/O automaton $\mathcal{S} = (M, V_0, V_1, m_I, \Rightarrow)$, where M is a finite set of (*memory*) *states*; V_0, V_1 are *input* and *output alphabets* respectively; $m_I \in M$ is the *initial state*; and $\Rightarrow \subseteq M \times V_0 \times V_1 \times M$ is the *transition relation*. If $m \overset{v_0}{\underset{v_1}{\Rightarrow}} m'$ then $\mathcal{S}$ being in the state m and reading v_0 from the input, outputs v_1 and changes its state to m'.

A play $\pi = \langle v_0, v_1, v_2, \dots \rangle$ is *consistent* with $\mathcal{S}$ if there is a sequence of memory states $\langle m_0, m_1, m_2, \dots \rangle$, such that $m_0 = m_I$ and $m_i \overset{v_{2i}}{\underset{v_{2i+1}}{\Longrightarrow}} m_{i+1}$ for every $i = 0, 1, 2, \dots$.

A strategy automaton $\mathcal{S}$ *realizes* a *strategy* ζ *with (finite) memory* M, defined as follows. For a finite play $\pi = \langle v_0, \dots, v_{2k} \rangle$ consistent with $\mathcal{S}$ we set $\zeta(\pi) = v$ if for the memory m_k obtained as above we have $m_k \overset{v_{2k}}{\underset{v}{\Longrightarrow}} m$ and $(v_{2k}, v) \in E$.

In case M is a one element set, we call the strategy *memoryless*.

3 Upper bound

In this section we are going to determine (Theorem 6) for every winning condition $\mathcal{F}$ the number $m_{\mathcal{F}}$ such that for every game with the winning condition $\mathcal{F}$ the memory of size $m_{\mathcal{F}}$ is sufficient for a winning strategy in this game.

Let $\mathcal{F} \subseteq \mathcal{P}(C)$ be a winning condition. Define $\mathcal{F} \upharpoonright D$ as the set $\{D' \in \mathcal{F} : D' \subseteq D\}$.

Definition 2 (Zielonka tree of a winning condition) We define the *Zielonka tree of* $\mathcal{F} \subseteq \mathcal{P}(C)$ (denoted by $\mathcal{Z}_{\mathcal{F},C}$) inductively:

1. If $C \notin \mathcal{F}$ then $\mathcal{Z}_{\mathcal{F},C} = \mathcal{Z}_{\overline{\mathcal{F}},C}$ where $\overline{\mathcal{F}} = \mathcal{P}(C) \setminus \mathcal{F}$.
2. If $C \in \mathcal{F}$ then the root of $\mathcal{Z}_{\mathcal{F},C}$ is labelled with C. Let $C_0, C_1, \dots, C_{k-1}$ be all the maximal sets in $\{X \notin \mathcal{F} : X \subseteq C\}$. Then we attach to the root, as its subtrees, the Zielonka trees of $\mathcal{F} \upharpoonright C_i$, i.e. $\mathcal{Z}_{\mathcal{F} \upharpoonright C_i, C_i}$, for $i = 0, 1, \dots, k-1$.

The *domain* of $\mathcal{Z}_{\mathcal{F},C}$ (denoted by $||\mathcal{Z}_{\mathcal{F},C}||$) is defined as the set of *nodes* of the tree. The Zielonka tree $\mathcal{Z}_{\mathcal{F},C}$ can be formally seen as the labelling function $\mathcal{C} : ||\mathcal{Z}_{\mathcal{F},C}|| \to \mathcal{P}(C)$ assigning to every node in the domain its label as defined above. In the sequel we will usually write $\mathcal{Z}_{\mathcal{F}}$ to denote $\mathcal{Z}_{\mathcal{F},C}$, if C is clear from the context.

A node of $\mathcal{Z}_{\mathcal{F}}$ is a *0-level node* if it is labelled with a set from $\mathcal{F}$. Otherwise it is a *1-level node*.

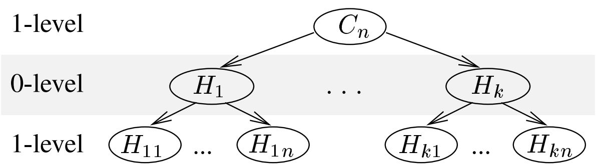


Figure 1. Example of a Zielonka tree

Example: Let $C_n = \{t_1, \dots, t_n, f_1, \dots, f_n\}$, for every $n \in \mathbb{N}$. For a boolean function $h : \{0,1\}^n \to \{0,1\}$ define a winning condition $\mathcal{F}_n \subseteq \mathcal{P}(C_n)$ as the family $\{H_1, \dots, H_k\} = \big\{\{x_1, \dots, x_n\} \subseteq C_n : h(\overline{x_1}, \dots, \overline{x_n}) = 0 \text{ and } x_i \in \{t_i, f_i\}\big\}$, where $\overline{t_i} = 1$ and $\overline{f_i} = 0$ for every $i = 1, \dots, n$. The root of the Zielonka tree $\mathcal{Z}_{\mathcal{F}_n}$ (c.f. Figure 1) is a 1-level node labelled with the set C_n of all colours. The k children of the root are 0-level nodes labelled with $H_1, \dots, H_k$ respectively. Every node labelled with $H_i = \{h_{i1}, \dots, h_{in}\}$ has n children labelled with the sets $H_{i1}, \dots, H_{in}$ respectively, where $H_{ij} = H_i \setminus \{h_{ij}\}$. $\square$

Let $\mathcal{A} = (V, V_0, V_1, E, C, \chi)$ be an arena. Now we will define the notion of an attractor. This will be a subarena of the arena. For simplicity in the sequel we will often identify subarenas with the sets of their vertices.

Definition 3 (Attractor) Define the *attractor for player* i *of a set* T *in the subarena* $W \subseteq V$ (denoted by $Attr_i^W(T)$) in the following way. Construct by transfinite induction the increasing sequence $T_0, T_1, T_2, \dots$ of sets of vertices:

- $T_0 = T \cap W$,
- $T_{\xi+1} = T_\xi \;\cup\; \{v \in V_i \cap W : vE \cap T_\xi \neq \emptyset\}$
  $\cup\; \{v \in V_{1-i} \cap W : vE \subseteq T_\xi\}$,
- if ξ is a limit ordinal then let $T_\xi = \bigcup_{\rho<\xi} T_\rho$.

Set $Attr_i^W(T) = \bigcup_\xi T_\xi$.

Remark: It is easy to show (see [15, 23, 20]) that $Attr_i^W(T)$ is the set of vertices in W from which player i can force the play into T using a memoryless strategy.

Let us consider a game $\mathcal{G} = (\mathcal{A}, \mathcal{F})$ with the winning condition $\mathcal{F} \subseteq \mathcal{P}(C)$ and the arena $\mathcal{A} = (V, V_0, V_1, E, C, \chi)$. We enrich the labels of the Zielonka tree $\mathcal{Z}_{\mathcal{F},C}$ to contain also sub-arenas of $\mathcal{A}$, on which player 0 has winning strategies in respective subgames. We also remove those subtrees for which the associated subarenas are empty. In this way we obtain the Zielonka tree of the game $\mathcal{G}$.

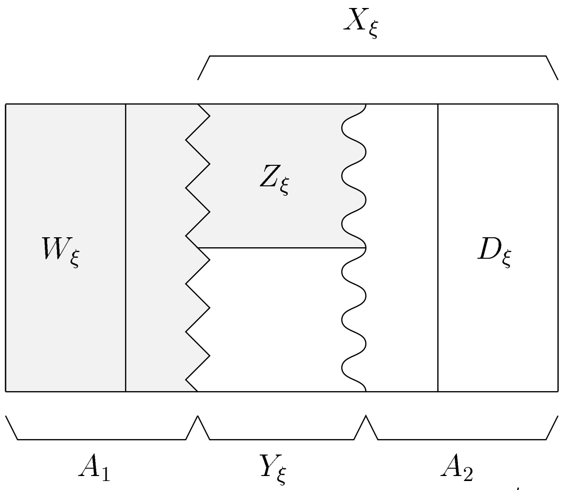
Definition 4 (Zielonka tree of a game) To define the *Zielonka tree* $\mathcal{Z}_{\mathcal{G}}$ *of the game* $\mathcal{G}$ we first define another labelling $\mathcal{V} : ||\mathcal{Z}_{\mathcal{F},C}|| \to \mathcal{P}(V)$ of the Zielonka tree $\mathcal{Z}_{\mathcal{F},C}$. The labelling $\mathcal{V}$ will assign to the nodes of $\mathcal{Z}_{\mathcal{F},C}$ subarenas of V. Hence eventually we will have two labelling functions: $\mathcal{C}$ as in the definition of $\mathcal{Z}_{\mathcal{F},C}$, and $\mathcal{V}$ which we are going to define now.

1. Set $\mathcal{V}(s) = V$ if s is the root node, i.e., the root is labelled with the set V of all vertices of the arena.
2. Suppose that for a node s' the function $\mathcal{V}(s') = V'$ is already defined and $\mathcal{C}(s') = C' \in \mathcal{F}$. For every son s'' of s' we set $\mathcal{V}(s'') = V' \setminus Attr_0^{V'}(\chi^{-1}(C' \setminus \mathcal{C}(s'')))$.
3. Suppose that $\mathcal{V}(s') = V'$ is already defined and $\mathcal{C}(s') = C' \notin \mathcal{F}$. Let $s_0, s_1, \dots, s_{k-1}$ be all the sons of s' in $\mathcal{Z}_{\mathcal{F},C}$, and $\mathcal{C}(s_i) = C_i$ for $i = 0, 1, \dots, k-1$. Set $\mathcal{V}(s_i) = U_i$ where the sets U_i are computed as follows:

   Set initially $W_0 := \emptyset$ and iterate through $\xi = 0, 1, 2, \dots$ until $W_\xi = W_{\xi+1} = \dots = W_{\xi+k-1}$. The sequence $\langle W_\xi \rangle$ is defined by transfinite induction. If ξ is a limit ordinal then we set $W_\xi = \bigcup_{\rho<\xi} W_\rho$, and otherwise perform the following computations (see Figure 2):

   (a) Set $X_\xi = V' \setminus Attr_0^{V'}(W_\xi)$.

   (b) Set $Y_\xi = X^\xi \setminus Attr_1^{X_\xi}(\chi^{-1}(C' \setminus C_{(\xi \bmod k)}))$.



The shaded region is equal to $W_{\xi+1}$; $A_1 = Attr_0^{V'}(W_\xi)$; $D_\xi = \chi^{-1}(C' \setminus C_{(\xi \bmod k)})$; $A_2 = Attr_1^{X_\xi}(D_\xi)$.

Figure 2. Calculating $W_{\xi+1}$

(c) Take Z_ξ to be the winning set for player 0 in the game $(\mathcal{A} \upharpoonright Y^\xi, \mathcal{F} \upharpoonright C_{(\xi \bmod k)})$.

(d) Set $W_{\xi+1} = Attr_0^{V'}(W_\xi) \cup Z_\xi$.

Eventually define $U_i = \bigcup_\xi \{Z_\xi : \xi \bmod k = i\}$.

The *domain* of the Zielonka tree $\mathcal{Z}_\mathcal{G}$ is defined as $||\mathcal{Z}_\mathcal{G}|| = \{s \in ||\mathcal{Z}_{\mathcal{F},C}|| : \mathcal{V}(s) \neq \emptyset\}$, i.e., the set of nodes labelled by $\mathcal{V}$ with a nonempty subarena of V. Each node s of $\mathcal{Z}_\mathcal{G}$ is labelled with the pair $(\mathcal{C}(s), \mathcal{V}(s))$.

Definition 5 (Memory of a Zielonka tree) Let $\mathcal{Z}$ be a Zielonka tree. A subtree of $\mathcal{Z}$ is called a *1-subtree*, if it can be obtained from $\mathcal{Z}$ by leaving at most one child of every 1-level node. We define the *memory of* $\mathcal{Z}$ to be the set $M_\mathcal{Z} = \{1, 2, \ldots, m_\mathcal{Z}\}$, where $m_\mathcal{Z}$ is the maximal number of leaves of a 1-subtree of $\mathcal{Z}$. We will use $M_\mathcal{F}$ and $m_\mathcal{F}$ to denote the memory and the size of memory of the Zielonka tree $\mathcal{Z}_\mathcal{F}$. Similarly we will use $M_\mathcal{G}$ and $m_\mathcal{G}$ when referring to the memory and the size of memory of the tree $\mathcal{Z}_\mathcal{G}$. Clearly, for a game $\mathcal{G} = (\mathcal{A}, \mathcal{F})$ we have that $||\mathcal{Z}_\mathcal{G}|| \subseteq ||\mathcal{Z}_\mathcal{F}||$, hence $m_\mathcal{G} \leq m_\mathcal{F}$.

Theorem 6 (Upper bound)
For every game $\mathcal{G}$ with the winning condition $\mathcal{F}$, player 0 has a strategy $\zeta_\mathcal{G}$ winning from every node of her winning set and requiring memory of size at most $m_\mathcal{F}$.

For the rest of this section let us fix a game $\mathcal{G} = (\mathcal{A}, \mathcal{F})$ with the arena $\mathcal{A} = (V, V_0, V_1, E, C, \chi)$. To prove the theorem we will construct an I/O automaton realizing a strategy in this game. This automaton will have $m_\mathcal{G}$ states. First we need some definitions.

We start with the definition of a numbering of the leaves of $\mathcal{Z}_\mathcal{G}$. Let $\mathcal{Z}$ be a Zielonka tree and $s \in \mathbb{N}$. We define the *s-numbering of* $\mathcal{Z}$ inductively:

1. If the root of $\mathcal{Z}$ is a leaf, then let s be its number.
2. If the root of $\mathcal{Z}$ is a 0-level node, then let $\mathcal{Z}_0, \ldots, \mathcal{Z}_{k-1}$ be all its immediate subtrees. The s-numbering of $\mathcal{Z}$ is obtained by the s_i-numbering of $\mathcal{Z}_i$ for every $i = 0, \ldots, k-1$, where $s_i = s + \sum_{j=0}^{i-1} m_{\mathcal{Z}_j}$.
3. If the root of $\mathcal{Z}$ is a 1-level node, then let $\mathcal{Z}_0, \ldots, \mathcal{Z}_{k-1}$ be all its immediate subtrees. The s-numbering of $\mathcal{Z}$ is obtained by the s-numbering of $\mathcal{Z}_i$ for every $i = 0, \ldots, k-1$.

The numbering of the leaves of $\mathcal{Z}_\mathcal{G}$ we are after is simply the 1-numbering of $\mathcal{Z}_\mathcal{G}$. Observe, that this numbering associates with every leaf of the Zielonka tree $\mathcal{Z}_\mathcal{G}$ an element of $M_\mathcal{G}$.

A memory $m \in M_\mathcal{G}$ does not uniquely determine a path from the root down the Zielonka tree $\mathcal{Z}_\mathcal{G}$. The reason is that a memory specifies only branchings at 0-level nodes of the tree, i.e., at a 0-level node we branch into the only subtree that may possibly contain leaves numbered with m. A pair $(v, m) \in V \times M_\mathcal{G}$, however, is just enough to single out such a unique path. To determine the "missing" branchings at 1-level nodes of $\mathcal{Z}_\mathcal{G}$, we use the fact, that sets of vertices labelling the children of a 1-level node of $\mathcal{Z}_\mathcal{G}$ are pairwise disjoint. Thus either we branch to the unique child of a 1-level node which is labelled with a set of vertices containing v, or we have reached the end of the path if none such a child exists.

Definition 7 (Anchor node) We will say that $s \in ||\mathcal{Z}_\mathcal{G}||$ is the *anchor node for* $(v, m) \in V \times M_\mathcal{G}$, if it is the lowest node on the unique path for (v, m), as described above, such that v belongs to the set of vertices labelling s in the Zielonka tree $\mathcal{Z}_\mathcal{G}$.

Before we describe the strategy for player 0 in the game $\mathcal{G}$, we shall define the *memory update function* and the *next move function*, which will be the essential components of the strategy.

Definition 8 (Memory update function) We define the *memory update function* $\rho_\mathcal{G} : V \times M_\mathcal{G} \to M_\mathcal{G}$. Given $(v, m) \in V \times M_\mathcal{G}$ we first find the anchor node $s \in ||\mathcal{Z}_\mathcal{G}||$ for (v, m). We define $\rho_\mathcal{G}(v, m)$ by cases:

- If s is a leaf in $||\mathcal{Z}_\mathcal{G}||$ or s is a 1-level node then $\rho_\mathcal{G}(v, m) = m$.
- Otherwise s is a 0-level node. Find the son s' of s, with a leaf numbered m in the subtree rooted in s'. Suppose it is the i-th son of s.

– If $v \in \chi^{-1}(\mathcal{C}(s) \setminus \mathcal{C}(s'))$ then $\rho_{\mathcal{G}}(v, m) = m'$, where $m' \in M_{\mathcal{G}}$ is the lowest number associated with a leaf in the $((i+1) \bmod k)$-th subtree of s, and k is the number of children of s in $\mathcal{Z}_{\mathcal{G}}$.

– Otherwise $\rho_{\mathcal{G}}(v, m) = m$.

Definition 9 (Next move function) We define the *next move function* $\sigma_{\mathcal{G}} : V_0 \times M_{\mathcal{G}} \to V_1$. Let $(v, m) \in V \times M_{\mathcal{G}}$ and let s' be the anchor node for (v, m). Let (C', V') be the label of s'. We define $\sigma_{\mathcal{G}}(v, m)$ by cases:

1. If s' is a leaf in $||\mathcal{Z}_{\mathcal{G}}||$ then $\sigma_{\mathcal{G}}(v, m) = v'$ for some $v' \in V \cap vE$.

2. If s' is a 0-level node, then find the son s'' of s', with a leaf numbered m in the subtree rooted in s''. Let (C'', V'') be the label of s'' in the Zielonka tree $\mathcal{Z}_{\mathcal{G}}$.

   (a) If $v \in \chi^{-1}(C' \setminus C'')$ then $\sigma_{\mathcal{G}}(v, m) = v'$ for some $v' \in V \cap vE$.

   (b) Otherwise $\sigma_{\mathcal{G}}(v, m) = \tau(v)$ where $\tau : V_0 \rightharpoonup V_1$ is the memoryless strategy "attract to the set $\chi^{-1}(C' \setminus C'')$".

3. If s' is a 1-level node and $\mu = \max\{\xi : v \notin W_\xi\}$ (where W_ξ for $\xi = 0, 1, 2, \ldots$ are the same as calculated in the definition of $\mathcal{V}(s')$), then $\sigma_{\mathcal{G}}(v, m) = \tau(v)$ where $\tau : V_0 \rightharpoonup V_1$ is the memoryless strategy "attract to the set W_μ".

Remark: According to our proviso, player 0 has a strategy from every vertex of the game. With this assumption it can be shown (see [23]) that indeed $v \in \bigcup_\xi W_\xi$; hence there exists μ required in the last clause of the above definition.

Definition 10 (The strategy $\zeta_{\mathcal{G}}$) The strategy $\zeta_{\mathcal{G}}$ in the game $\mathcal{G}$ is realized by the I/O automaton $\mathcal{S} = \langle M_{\mathcal{G}}, V_0, V_1, m_I, \Rightarrow \rangle$, where $m_I = 1 \in M_{\mathcal{G}}$ and $m \overset{v}{\underset{v'}{\Rightarrow}} m'$ whenever v', m' can be "computed" by performing the following actions:

- update the memory according to the move of player 1 ($m'' = \rho(v, m)$),
- compute the move to be performed ($v' = \sigma(v, m'')$),
- update the memory according to this move ($m' = \rho(v', m'')$).

The strategy $\zeta_{\mathcal{G}}$ uses memory $M_{\mathcal{G}}$ of size $m_{\mathcal{G}} \le m_{\mathcal{F}}$ (see Definition 5). To prove Theorem 6 it remains to show that $\zeta_{\mathcal{G}}$ is a winning strategy for player 0.

Lemma 11 The strategy $\zeta_{\mathcal{G}}$ is winning for player 0.

Proof
Consider an infinite play $\pi = \langle v_1, v_2, v_3, \ldots \rangle$ consistent with the strategy $\zeta_{\mathcal{G}}$ and let $\langle m_1, m_2, m_3, \ldots \rangle$ be the sequence of corresponding memories, i.e., we have $v_{2j} = \sigma_{\mathcal{G}}(v_{2j-1}, m_{2j-1})$ and $m_j = \rho_{\mathcal{G}}(v_j, m_{j-1})$, for every $j = 1, 2, \ldots$, where $m_0 = m_I \in M_{\mathcal{G}}$ is the initial memory of $\zeta_{\mathcal{G}}$.

Let $\langle s_1, s_2, s_3, \ldots \rangle$ be the sequence of the anchor nodes for the respective pairs (v_j, m_j). Define s_π to be the root of the lowest subtree of $\mathcal{Z}_{\mathcal{G}}$ containing almost every s_j. Let (C', V') be the label of s_π in $\mathcal{Z}_{\mathcal{G}}$. Let (C_i, V_i), for $i = 0, \ldots, k-1$, be the labels of the k children of s_π.

Observation 11.1 s_π is a 0-level node.

Proof: Suppose s_π is a 1-level node. Let Z_ξ for $\xi = 0, 1, 2, \ldots$ be the same as calculated in the definition of $\mathcal{V}(s_\pi)$. Every time s_π is the anchor node for some (v_j, m_j) the strategy $\zeta_{\mathcal{G}}$ attracts the play to a set Z_ξ with a smaller ξ than before. We can, however, decrease ξ only finitely many times, so almost every v_j belongs to some fixed $Z_\xi \subseteq V_{(\xi \bmod k)}$. Thus almost every s_j belongs to the subtree of $\mathcal{Z}_{\mathcal{G}}$ rooted at $(\xi \bmod k)$-th child of s_π. This, however, contradicts the definition of s_π. $\square$

Observation 11.2 s_π occurs infinitely often in $\langle s_1, s_2, s_3, \ldots \rangle$.

Proof: If it did not, then there would exist an i, such that for almost every j we would have $v_j \in V_i$. This contradicts the definition of s_π. $\square$

From the definition of s_π it is evident that $\mathit{Inf}(\pi) \subseteq C'$. From Observations 11.1 and 11.2 it follows that for every $i = 0, \ldots, k-1$ the strategy $\zeta_{\mathcal{G}}$ attracts the play π to the set $\chi^{-1}(C' \setminus C_i)$ infinitely many times, so in fact we have that $\mathit{Inf}(\pi) \not\subseteq C_i$.

As, by Observation 11.1, s is a 0-level node, we have that $\mathit{Inf}(\pi) \in \mathcal{F}$. Hence $\zeta_{\mathcal{G}}$ is winning for player 0 and Theorem 6 is proved. $\square$

Remark: Note that, in particular, if $\mathcal{P}(C) \setminus \mathcal{F}$ is closed under union, i.e., $\mathcal{F}$ can be expressed by a *Rabin condition* (see [23]), then the strategy $\zeta_{\mathcal{G}}$ is memoryless. Memoryless determinacy for such games was shown in [7, 13].

Remark: Observe that every 1-subtree of a tree from the Example on page 4 (Figure 1) has n leaves (labelled with $H_{i1}, \ldots, H_{in}$ for some i). From Theorem 6 it follows that winning strategies for player 0 in every game with such a winning condition need at most memory of size $\mathcal{O}(n)$.

Corollary 12 (Complexity of games) The problem $\{(\mathcal{Z}_{\mathcal{F}}, \mathcal{A}, v_0)$: player 0 has a winning strategy in the game $(\mathcal{A}, \mathcal{F})$ starting from the vertex v_0 of the arena $\mathcal{A}\}$ is in NP $\cap$ co-NP.

Proof
Let $\mathcal{G} = (\mathcal{A}, \mathcal{F})$. If player 0 has a winning strategy in $\mathcal{G}$ then the NP-witness is the strategy $\zeta_{\mathcal{G}}$. To prove that the problem is in NP it suffices to show, that there is a polynomial time algorithm to decide if this strategy is winning for player 0.

Let $G = (V \times M_{\mathcal{F}}, E_G)$ be the bipartite graph, such that $\big((v, m), (v', m')\big) \in E_G$ if either:

- $m = m'$ and $(v, v') \in V_1 \times V_0$ is an edge of the arena $\mathcal{A}$, or
- $m \underset{v'}{\overset{v}{\Rightarrow}} m'$ holds for the I/O automaton realizing $\zeta_{\mathcal{G}}$.

We restrict G to the vertices reachable from the vertex (v_0, m_I), where m_I is the initial memory of $\zeta_{\mathcal{G}}$. Let the colouring of the nodes of G be inherited from the colouring of the arena $\mathcal{A}$, i.e., $\chi((v, m)) = \chi(v)$. Deciding if the strategy $\zeta_{\mathcal{G}}$ is winning for player 0 is clearly equivalent to checking that all the infinite paths in the graph G are coloured with sets belonging to $\mathcal{F}$.

Now we give a short description of an algorithm to perform this task.

Divide G into strongly connected components (s.c.c.). If for some s.c.c. H it holds that $\vec{\chi}(H) \notin \mathcal{F}$, then clearly $\zeta_{\mathcal{G}}$ is not winning for player 0.

Otherwise for every s.c.c. H of G find all 0-level nodes $s \in ||\mathcal{Z}_{\mathcal{F}}||$, such that $\vec{\chi}(H) \subseteq \mathcal{C}(s)$, but the inclusion does not hold for any of the grandchildren of s. For every such s let $s_1, s_2, \ldots, s_k$ be its children in $\mathcal{Z}_{\mathcal{F}}$. For every $i = 1, 2, \ldots, k$ check recursively if $\Big(G \restriction \chi^{-1}\big(\mathcal{C}(s_i)\big)\Big)$ satisfies $\mathcal{F} \restriction \mathcal{C}(s_i)$. $\square$

Remark: Corollary 12 can be seen as a generalisation of the result of Emerson, Jutla and Sistla [10] that the decision problem for games with *Rabin chain (parity) conditions* is in NP ∩ co-NP. In this special case the size of the Zielonka tree is linear in the number of colours.

By a careful analysis of the strategy synthesis algorithm implicit in the definition of the strategy $\zeta_{\mathcal{G}}$ one can obtain the following fact.

Corollary 13 (Strategy synthesis algorithm) There is an algorithm that given a game $\mathcal{G} = (\mathcal{A}, \mathcal{F})$ computes a winning strategy for player 0 in time $\mathcal{O}((|\mathcal{A}| * d)^h)$; here d is the maximal degree of a node and h is the height of the Zielonka tree $\mathcal{Z}_{\mathcal{F}}$.

4 Lower bounds

4.1 The $m_{\mathcal{F}}$ lower bound

Here we consider the lower bound on the size of memory needed by a winning strategy in case when the size of a graph is not limited by the number of colours. We show that in this case the upper bound of Theorem 6 is tight.

Theorem 14 (Lower bound)
For every winning condition $\mathcal{F}$ there is an arena $\mathcal{A}_{\mathcal{F}}$ such that every winning strategy for player 0 in the game $(\mathcal{A}_{\mathcal{F}}, \mathcal{F})$ requires memory of size at least $m_{\mathcal{F}}$.

Proof
Let $\mathcal{F} \subseteq \mathcal{P}(C)$. Assume that $C \in \mathcal{F}$. Otherwise we take a son s' of the root, such that, there is a 1-subtree with $m_{\mathcal{F}}$ leaves below s'. Then we can take the label C' of s' and continue our construction for $\mathcal{F} \restriction C'$ instead of $\mathcal{F}$.

Let T be a 1-subtree of the Zielonka tree $\mathcal{Z}_{\mathcal{F}}$ with $m_{\mathcal{F}}$ leaves and let $\sigma_1, \ldots, \sigma_{m_{\mathcal{F}}}$ be all the leaves of T. We define the arena $\mathcal{A}_{\mathcal{F}}$ as follows. For every set of colours $E = \{e_1, \ldots, e_l\}$ we define a subarena $\mathtt{Pick}(E)$ as on Figure 3(a). In the square (uncoloured) vertex player 1 makes a choice to visit a vertex coloured with one of the colours in E. To facilitate the description of the whole arena let us use the sequence of labels on the path leading to a node of T to identify this node. A notation $C_1D_1 \ldots C_nD_n(C_{n+1})$ denotes a leaf to which leads a path with labels $C_1D_1 \ldots C_nD_n$ or $C_1D_1 \ldots C_nD_nC_{n+1}$. For every leaf $\sigma = C_1D_1 \ldots C_nD_n(C_{n+1})$ of T we define a subarena $\mathtt{Box}(\sigma)$ as on Figure 3(b). Again, in the square (uncoloured) vertex player 1 makes a move. Finally the arena $\mathcal{A}_{\mathcal{F}}$ is defined as on Figure 3(c), where in the circle (uncoloured) vertex player 0 makes a move.

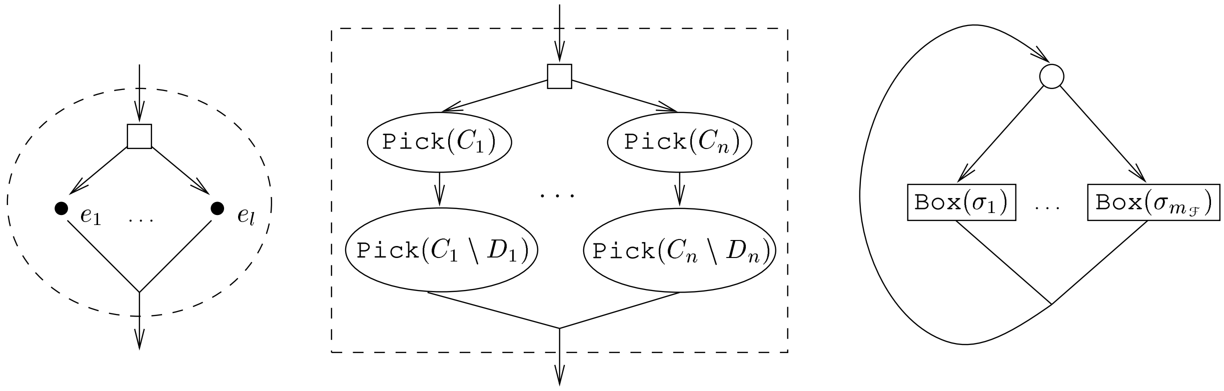
Observe that a play on the arena $\mathcal{A}_{\mathcal{F}}$ consists of an infinite sequence of the following choices:

1. Player 0 chooses $\mathtt{Box}(\sigma)$ for some leaf $\sigma = C_1D_1 \ldots C_nD_n(C_{n+1})$ of T.
2. Player 1 chooses one of the indices $j \in \{1, \ldots, n\}$.
3. Player 1 picks a vertex coloured with one of the colours in C_j and then a vertex coloured with one of the colours in $C_j \setminus D_j$.

Claim 14.1 Player 0 has a winning strategy in the game $(\mathcal{A}_{\mathcal{F}}, \mathcal{F})$.

Proof: The only significant choices of player 0 are made in step 1. In the strategy we are defining her choice will depend on the choice made by player 1 in step 2 of the preceding round of the play. Suppose that player 0 has chosen $\mathtt{Box}(\sigma)$ and player 1 has chosen the index j in step 2. Then in the next round player 0 chooses $\mathtt{Box}(\sigma')$, where σ' is some leaf in the subtree of T rooted in the next child of the node $C_1D_1 \ldots C_j$, after the child $C_1D_1 \ldots C_jD_j$. (If $C_1D_1 \ldots C_jD_j$ is the last child of its parent then pick a leaf σ in the subtree rooted in the first child of $C_1D_1 \ldots C_j$.)

Now we will argue that the strategy just described is winning for player 0. Consider a play π consistent with the strategy. Let $\gamma = C_1D_1 \ldots C_i$ be the root of the lowest subtree of T containing all the leaves of T chosen infinitely



(a) (b) (c)

Figure 3. The arena $\mathcal{A}_{\mathcal{F}}$.

often by player 0 in step 1. Let $\gamma_j = C_1 D_1 \dots C_i D_{i_j}$ for $j = 1, \dots, k$ be all the children of γ in T. Observe that for every $j = 1, \dots, k$ there is a leaf σ_j in the subtree rooted in γ_j, such that the index i is chosen by player 1 in step 2 infinitely often in $\texttt{Box}(\sigma_j)$. This, however, implies that $\mathit{Inf}(\pi) \not\subseteq D_{i_j}$ for every $j = 1, \dots, k$. On the other hand it is not hard to see that $\mathit{Inf}(\pi) \subseteq C_i$, hence $\mathit{Inf}(\pi) \in \mathcal{F}$. $\Box$

If the size of the memory of a strategy for player 0 in the game $(\mathcal{A}_{\mathcal{F}}, \mathcal{F})$ is less than $m_{\mathcal{F}}$, then clearly there is a leaf σ of T, such that player 0 playing according to this strategy never visits $\texttt{Box}(\sigma)$. Hence, the following claim establishes the lower bound of the theorem.

Claim 14.2 Consider a strategy for player 0 in the game $(\mathcal{A}_{\mathcal{F}}, \mathcal{F})$ and a leaf σ in the tree T. If player 0 never visits $\texttt{Box}(\sigma)$ when playing according to the strategy, then the strategy is not winning for player 0.

Proof: Let $\sigma = C_1 D_1 \dots C_n D_n (C_{n+1})$. We will now provide a counter-strategy for player 1 allowing him to make one of D_i's the set of frequent colours. Which D_i it will turn out to be depends on the actual behaviour of the strategy of player 0.

To play according to his counter-strategy player 1 keeps in his memory a tuple $\vec{d} \in D_1 \times \cdots \times D_n$ to remember which element of each D_i is currently his target.

Assume that player 0 chooses $\texttt{Box}(\delta)$ such that $\delta = C_1 D_1 \dots C_j D'_j \dots$ with $D'_j \neq D_j$, i.e., j is the smallest index at which σ and δ differ. Player 1 responds making the following choices in $\texttt{Box}(\delta)$:

1. Choose $\texttt{Pick}(C_j)$ and then pick a vertex coloured with d_j, where $(d_1, \dots, d_n)$ is the current memory of player 1.

2. From $\texttt{Pick}(C_j \setminus D'_j)$ choose a vertex coloured with some $c \in D_j$. It is possible to do so, because $D_j \subset C_j$ and $D_j \setminus D'_j \neq \emptyset$.

After performing these moves player 1 updates his current memory $(d_1, \dots, d_n)$ by changing its j-th component to the next element of D_j in some fixed linear order (or to the minimal element of D_j in this order, if d_j was maximal).

Now we will argue that this counter-strategy guarantees a win for player 1. For a play π formed according to this counter-strategy consider the set I of indices j chosen infinitely often in the way described above. Set $i = \min(I)$ and observe that from some moment of the play π only colours in the set $D_i \cup D_{i+1} \cup \cdots \cup D_n = D_i$ appear (recall that $D_l \supset D_{l+1}$ for $l = 1, \dots, n-1$). Moreover, each colour of D_i is hit infinitely often. Thus $\mathit{Inf}(\pi) = D_i$. This completes the proof since D_i is a winning set for player 1.
$\Box$ $\Box$

Remark: The arena $\mathcal{A}_{\mathcal{F}}$ constructed in the proof of Theorem 14 is roughly of size $m_{\mathcal{F}}$. Hence, if for a family of conditions $\langle \mathcal{F}_n \rangle_{n \in \mathbb{N}}$ the number $m_{\mathcal{F}_n}$ grows superpolynomially in the number of colours of $\mathcal{F}_n$, then our lower bound does not apply to the case when we restrict ourselves to arenas of size polynomial in the number of colours. We do not know whether the $m_{\mathcal{F}}$ bound is strict in this case.

4.2 Optimality of the LAR's

Now we show a factorial lower bound establishing optimality of the LAR data structure even for games with arena sizes bounded linearly in the number of colours of the winning condition. Optimality of LAR's in the case when we do not bound the size of the game follows from the previous section.

Theorem 15 (LAR's are optimal)
There is a family of games $\langle \mathcal{G}_n \rangle_{n \in \mathbb{N}}$, such that the arena of $\mathcal{G}_n$ is of size $\mathcal{O}(n)$ and every winning strategy for player 0 in $\mathcal{G}_n$ has memory of size at least $n!$.

Proof
The game $\mathcal{G}_n$ has the arena $\mathcal{A}_n = (V, V_0, V_1, E, C_n, \chi)$, where: $V_0 = \{-n, \ldots, -1\}$, $V_1 = \{1, \ldots, n\}$, $E = \{(-i, j), (i, -j) : i, j \in \{1, \ldots, n\}\}$ and $C_n = V$. We identify vertices with colours setting $\chi(v) = v$, for every $v \in V$. The winning condition of $\mathcal{G}_n$ is defined as $\mathcal{F}_n = \{C \subseteq C_n : |C \cap V_0| = \max(C \cap V_1)\}$.

It is not difficult to see that player 0 has an LAR winning strategy in $\mathcal{G}_n$ (in fact it suffices to consider LAR's over colours from the set V_0 only).

Suppose that $\mathcal{S} = (M, V_0, V_1, m_I, \Rightarrow)$ is an I/O automaton realizing a strategy for player 0. For the sake of this proof by the *transition graph of* $\mathcal{S}$ we will understand the graph $(M, \rightarrow)$ with edges labelled by elements of V_0, where $m \overset{v_0}{\rightarrow} m'$ if there is a $v_1 \in V_1$, such that $m \underset{v_1}{\overset{v_0}{\Rightarrow}} m'$ holds.

In the following lemma we slightly strengthen the statement of the theorem, in order for the proof by induction on n to go through.

Lemma 16 Let $\mathcal{S}$ be the I/O automaton realizing a winning strategy for player 0 in $\mathcal{G}_n$. Then there is a state $m \in M$ reachable from the initial state of the transition graph of $\mathcal{S}$, and a finite sequence $w \in V_0^*$ with at least one occurrence of every element of V_0, such that:

1. $m \overset{w}{\rightarrow}_* m$, i.e., the path in the transition graph of the automaton $\mathcal{S}$, labelled with the input word w and starting in state m is a loop,
2. $|M_w| \geq n!$, where M_w is the set of states of $\mathcal{S}$ on the path described above.

Proof: W.l.o.g. we can assume that the transition graph of the automaton $\mathcal{S}$ is strongly connected. Otherwise we could restrict the transition graph of $\mathcal{S}$ to one of its strongly connected components (s.c.c.), that is reachable from the initial state, and is final in the acyclic graph of all s.c.c.'s of $\mathcal{S}$ (i.e., there are no edges going out of this s.c.c. in the transition graph of $\mathcal{S}$). As player 1 can in a finite number of moves force the play into a reachable final s.c.c., clearly after such a restriction the remaining strategy is still winning for player 0.

The base case of the induction follows immediately from the assumption that the transition graph of $\mathcal{S}$ is strongly connected.

Let $\mathcal{H}_1, \ldots, \mathcal{H}_n$ be sub-games of $\mathcal{G}_n$, such that $\mathcal{H}_i$ is obtained by removing $-i$ from V_0 and n from V_1. It is easy to see that I/O automata $\mathcal{S}_1, \ldots, \mathcal{S}_n$ obtained by straightforward restrictions[1] of $\mathcal{S}$ realize winning strategies for player 0 in games $\mathcal{H}_1, \ldots, \mathcal{H}_n$. From the induction hypothesis it follows that there are states $m_1, \ldots, m_n \in M$, and finite sequences $w_1, \ldots, w_n$, such that $w_i \in (V_0 \setminus \{-i\})^*$ and $|M_i| \geq (n-1)!$, where M_i is the set of states on the path $m_i \overset{w_i}{\rightarrow}_* m_i$.

Observe that the outputs produced by $\mathcal{S}$ on any path $m_i \overset{w_i}{\rightarrow}_* m_i$ for $i = 1, \ldots, n$ do not contain any occurrence of $n \in V_1$. Otherwise, the infinite input sequence $(w_i)^\omega$ would form a play winning for player 1. This, however, would contradict the assumption that $\mathcal{S}$ realized a winning strategy for player 0 in $\mathcal{G}_n$.

We will now argue, that if $i \neq j$ then $M_i \cap M_j = \emptyset$. This would immediately imply that $\left|\bigcup_{i=1}^n M_i\right| \geq n * (n-1) = n!$. Assume the contrary, i.e., that there is a state $m \in M_i \cap M_j$ for some $i \neq j$. Then there are partitions $w_i = u_1 \cdot u_2$ and $w_j = v_1 \cdot v_2$, such that $m_i \overset{u_1}{\rightarrow}_* m \overset{u_2}{\rightarrow}_* m_i$ and $m_j \overset{v_1}{\rightarrow}_* m \overset{v_2}{\rightarrow}_* m_j$. So there is a loop $m \overset{u_2 \cdot u_1 \cdot v_2 \cdot v_1}{\longrightarrow_*} m$. Hence, the infinite input sequence $(u_2 \cdot u_1 \cdot v_2 \cdot v_1)^\omega$ induces a play winning for player 1, because the sequence $u_2 \cdot u_1 \cdot v_2 \cdot v_1$ contains occurrences of all n elements of V_0, and the output generated by $\mathcal{S}$ does not contain a occurrence of $n \in V_1$. This, however, contradicts the assumption that $\mathcal{S}$ realized a winning strategy for player 0.

To finish the proof of the Lemma it remains to show that the sequences w_i for $i = 1, \ldots, n$ can be composed to yield a sequence w, such that $m \overset{w}{\rightarrow}_* m$ for some $m \in M$ and this cycle includes all cycles $m_i \overset{w_i}{\rightarrow}_* m_i$ as its sub-paths. Again, the assumption of strong connectedness of the transition graph of $\mathcal{S}$ makes this task trivial. $\square$ $\square$

5 Polynomially representable strategies

The lower bound of the previous section indicates that the size of the memory of I/O automata realizing winning strategies may be inherently big as a function of the size of the arena. Here we consider a representation of strategies in terms of so called *p-automata*, which are parallel compositions of I/O automata. It turns out that p-automata are capable of encoding strategies in a more succinct way than I/O automata.

One would like to measure the size of a winning strategy in a game in terms of two parameters: the size of the arena and the size of the winning condition. From the perspective of program synthesis it seems natural to assume that the arena is given just as a graph. On the other hand it is much too restrictive to assume that winning conditions are given in Muller form. For example, Rabin conditions can be sometimes exponentially more succinct than Muller conditions. To avoid these problems of different representations we propose to represent winning conditions by means of Turing machines, or equivalently, by p-automata. For a function $p(n)$, we will say that a family $\mathcal{M}$ of winning conditions is $p(n)$-recognisable if for every winning condition $\mathcal{F} \in \mathcal{M}$ with n colours there exists a p-automaton of size

[1] More precisely, in order to obtain $\mathcal{S}_i$ we restrict the input alphabet of $\mathcal{S}$ to $V_0 \setminus \{-i\}$, and change the output component of the transition relation whenever it is n to, for example, $(n-1)$.

$p(n)$ recognizing $\mathcal{F}$. Hence our notion of recognizability will be a non-uniform one. We find it more intuitive to express this notion in terms of p-automata.

The main result of this section is that if a family of winning conditions is $p(n)$-recognisable for some polynomial $p(n)$ then winning strategies in games with these conditions are representable by polynomial size p-automata. We also show that it is difficult to characterize all families of winning conditions admitting polynomial size strategies in terms of p-automata. For this we recall that it is an open problem to show an example of a Boolean function not computable in nonuniform NC^1. Every family of conditions not admitting polynomial size strategies would give an example of such a function. Hence, although there are many families of winning conditions that do not admit polynomial size p-strategies, it is not easy to come across one of them.

Definition 17 (p-automaton) Let $\langle \mathcal{R}_i \rangle_{i=1}^n$ be a tuple of I/O automata, each of the form $\mathcal{R}_i = (Q_i, \Sigma_{in}, \Sigma_{out}, q_i, \Longrightarrow_i)$. We assume that there is a special letter $\tau \in \Sigma_{in} \cap \Sigma_{out}$. If $q \overset{\tau}{\underset{a}{\Longrightarrow}}_i q'$ then, intuitively, the automaton does not read from the input; similarly for $\overset{a}{\underset{\tau}{\Longrightarrow}}_i$ but then the automaton outputs nothing. A p-automaton is a sequence of I/O automata $\langle \mathcal{R}_i \rangle_{i=1}^n$ together with a set Σ_{ext} of *external letters*.

One can think of such an automaton as of a parallel composition of automata $\langle \mathcal{R}_i \rangle_{i=1}^n$ restricted on actions not in Σ_{ext}. In other words, it can be considered to be a CCS process: $(\mathcal{R}_1 || \dots || \mathcal{R}_n) \backslash ((\Sigma_{in} \cup \Sigma_{out}) - \Sigma_{ext})$. The semantics of p-automata is based on this intuition.

A sequence $\langle \mathcal{R}_i \rangle_{i=1}^n$ and a set Σ_{ext} as above define a global I/O automaton $\mathcal{R} = (Q_1 \times \dots \times Q_n, \Sigma_{in}, \Sigma_{out}, (q_1, \dots, q_n), \Longrightarrow_{\mathcal{R}})$ with the transition function defined by the rules:

$$\frac{q_i \overset{a}{\underset{b}{\Longrightarrow}}_{\mathcal{R}} q_i' \qquad a, b \in \Sigma_{ext}}{(\dots, q_i, \dots) \overset{a}{\underset{b}{\Longrightarrow}}_{\mathcal{R}} (\dots, q_i', \dots)}$$

$$\frac{q_i \overset{a}{\underset{\tau}{\Longrightarrow}}_{\mathcal{R}} q_i' \quad q_j \overset{\tau}{\underset{a}{\Longrightarrow}}_{\mathcal{R}} q_j' \qquad a \notin \Sigma_{ext}}{(\dots, q_i, \dots, q_j, \dots) \overset{\tau}{\underset{\tau}{\Longrightarrow}}_{\mathcal{R}} (\dots, q_i', \dots, q_j', \dots)}$$

We can use p-automata to succinctly represent languages over some alphabet Σ. For this we let $\Sigma_{ext} = \Sigma \cup \{tt, ff\}$ and say that a word $w \in \Sigma^*$ is accepted by a p-automaton if the global automaton outputs tt after reading w.

A *Muller sequence* $\langle \mathcal{M}_n \rangle_{n \in \mathbb{N}}$ is a sequence of sets of winning conditions, such that every $\mathcal{F} \in \mathcal{M}_n$ is a condition using at most n colours.

Definition 18 (Polynomially representable Muller sequence) We say that a Muller sequence $\langle \mathcal{M}_n \rangle_{n \in \mathbb{N}}$ *is polynomially representable* if there exists a polynomial $p(n)$, such that for every $\mathcal{F} \in \mathcal{M}_n$ there is a p-automaton of size $p(n)$ representing $\mathcal{F}$; here we consider $\mathcal{F}$ to be a set of sequences of colours.

We can also use p-automata to succinctly represent "sequential" I/O automata (as in Definition 1). Such an automaton $\mathcal{S}$, represented by $\mathcal{R}$, has all the same components as $\mathcal{R}$ but the transition relation. This relation is defined by: $\vec{q}_1 \overset{a}{\underset{b}{\Longrightarrow}}_{\mathcal{S}} \vec{q}_2$ if there are states $\vec{q}\,'$ and $\vec{q}\,''$, such that, $\vec{q}_1 \overset{a}{\underset{\tau}{\Longrightarrow}}_{\mathcal{R}} \vec{q}\,' \overset{\tau}{\underset{\tau}{\Longrightarrow}}{}^*_{\mathcal{R}} \vec{q}\,'' \overset{\tau}{\underset{b}{\Longrightarrow}}_{\mathcal{R}} \vec{q}_2$. In other words, we can read an input, do some number of τ moves, and then give an output.

Definition 19 (Polynomially representable strategies) We say that a Muller sequence $\langle \mathcal{M}_n \rangle_{n \in \mathbb{N}}$ *has polynomially representable strategies* if there exists a polynomial $p(n)$, such that for every game $(\mathcal{A}, \mathcal{F}_n)$ with $\mathcal{F}_n \in \mathcal{M}_n$, there is a parallel automaton of size $p(|\mathcal{A}|)$ representing a winning strategy for player 0 on her winning set.

Let us give an example of savings that can be made with this representation. Consider the game $\mathcal{G}_n$ as defined in the proof of Theorem 15. Player 0 has a winning strategy in this game that consists of keeping LAR memory and choosing a number equal to the number of negative integers appearing before the change pointer in the LAR (for a description of LAR memory see [20]). To implement this strategy using a p-automaton, we can take n I/O automata, each for storing one coordinate of the LAR. Then we add a component that takes care of input/output and of updating the other components. It should be obvious that the size of each of the components is linear in n. Hence there is a constant c, s.t., a winning strategy in $\mathcal{G}_n$ can be represented by a p-automaton of size cn. On the other hand Theorem 15 states that every "sequential" winning strategy for $\mathcal{G}_n$ must have size $\Omega(n!)$.

Not every game is so easy. A simple counting argument shows:

Proposition 20 There exists a Muller sequence $\langle \mathcal{M}_n \rangle_{n \in \mathbb{N}}$ that does not have polynomially representable strategies.

Next we will show that every polynomially representable Muller sequence has polynomially representable strategies and p-automata realizing these strategies can be constructed in polynomial time.

Proposition 21 There exists an algorithm that given a game $\mathcal{G}$ with the arena $\mathcal{A}$ and the winning condition represented by a p-automaton $\mathcal{W}$, constructs in time $\mathcal{O}(poly(|\mathcal{A}|, |\mathcal{W}|))$ a p-automaton $\mathcal{R}$ of size $\mathcal{O}(poly(|\mathcal{A}|, |\mathcal{W}|))$ realizing a winning strategy for player 0 on her winning set.

Proof
A careful inspection of the definition of the strategy $\zeta_{\mathcal{G}}$ assures that given an oracle for recognizing sets in the winning condition, the whole winning strategy can be in fact computed (however not necessarily output) in space polynomial in $\mathcal{A}$. The polynomial space Turing machine T computing the strategy can be in a straightforward way encoded by a p-automaton $\mathcal{T}$ of size linear in the size of configurations of T. Hence, to obtain the p-automaton $\mathcal{R}$ realizing the strategy $\zeta_{\mathcal{G}}$ it is enough to "compose" $\mathcal{T}$ and $\mathcal{W}$ so that the oracle questions of $\mathcal{T}$ are answered by $\mathcal{W}$. After "composition" with $\mathcal{W}$ we finally obtain a p-automaton of size $\mathcal{O}(poly(|\mathcal{A}|, |\mathcal{W}|))$.
$\square$

Remark: This proposition does not contradict the fact that, in general, constructing sequential strategy automata is difficult. It may be difficult to construct a sequential strategy automaton from the parallel one. The reason is that p-automaton may sometimes do a long sequence of τ-moves before answering the next move.

Corollary 22 Suppose that a family of winning conditions can be recognised by a Turing machine working in a space polynomial in the number of colours used in the condition. It follows that winning strategies in games with these conditions can be represented by polynomial size p-automata. In particular Rabin or Street winning conditions with the number of pairs polynomial in the number of colours admit polynomial size strategies.

It would be tempting to say that if a Muller sequence admits polynomially representable strategies then the sequence itself is polynomially representable. This turns out to be false.

Proposition 23 There exists a Muller sequence $\langle \mathcal{M}_n \rangle_{n\in\mathbb{N}}$ that has polynomially representable strategies, but is not polynomially representable.

Proof
Take a boolean function $h : \{0,1\}^* \to \{0,1\}$ that is not polynomially representable. Define the Muller sequence $\langle \mathcal{M}_n \rangle_{n\in\mathbb{N}}$ by $\mathcal{M}_{2n-1} = \emptyset$ and $\mathcal{M}_{2n} = \{\mathcal{F}_n\}$ for every $n = 1, 2, \ldots$, where $\mathcal{F}_n$ is as defined in the example on page 4. Clearly the Muller sequence $\langle \mathcal{M}_n \rangle_{n\in\mathbb{N}}$ is not polynomially representable. Nevertheless, from the Theorem 6 it follows that $\langle \mathcal{M}_n \rangle_{n\in\mathbb{N}}$ has polynomially representable strategies (see the second remark after the proof of Lemma 11). $\square$

The problem of characterizing families of winning conditions having polynomially representable strategies is at least as difficult as giving an answer to one of the main questions in nonuniform complexity theory. We will now show that if we had such a characterisation then we could show an example of a function $h : \{0,1\}^* \to \{0,1\}$ that is not computable in nonuniform NC^1. This is known to be the major open question about nonuniform complexity classes (see [2] pp. 763-764 for the description of the problem).

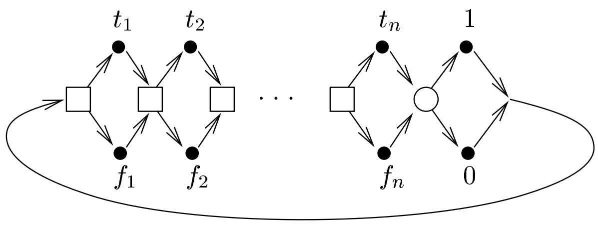


Figure 4. The arena of the game $G(h, n)$

We show that arbitrary characterisation of Muller sequences having polynomially representable strategies will allow us to decide whether a function is computable in PSPACE/poly.

Let $h : \{0,1\}^* \to \{0,1\}$ be a function. For every n construct a game $G(h, n)$ as in Figure 4. Player 1 moves in square vertices and player 0 in the circle vertex. The set of colours is $C_n = \{t_1, \ldots, t_n, f_1, \ldots, f_n, 0, 1\}$. The winning condition is the smallest set closed under union that contains the sets $\{x_1, \ldots, x_n, h(\overline{x}_1, \ldots, \overline{x}_n)\}$; where $x_i \in \{t_i, f_i\}$ and $\overline{t}_i = 1$, $\overline{f}_i = 0$, for every $i = 1, 2 \ldots, n$.

It should be clear that player 0 has a winning strategy in this game. The simplest one is to give correct answers each time. Intuitively also all other strategies should be "aware" of the correct values of h.

If h is computable in PSPACE/poly then, by Proposition 21, for every n there is a polynomially representable strategy in the game $G(h, n)$. If the strategies for all the games are polynomially representable then we claim that h is in PSPACE/poly. Let us take n and a p-automaton $\mathcal{R}_n$ of size $p(n)$ realizing a winning strategy in the game $G(h, n)$. To calculate the value of the function h on some arguments $x_1, \ldots, x_n$ we simulate a particular play in $G(h, n)$. In this play we make player 1 repeatedly choose $x_1, \ldots, x_n$ and see what are the answers of $\mathcal{R}_n$ in this play. The first answer may not be correct but eventually the answers of $\mathcal{R}_n$ must stabilize at the correct answer. This should happen after no more than d steps, where d is the number of configurations of $\mathcal{R}_n$. It should be clear that this simulation can be performed in PSPACE/poly.

6 Acknowledgements

The first two authors are grateful to Damian Niwiński for spiritual support and a wonderful introduction to the theory of automata on infinite objects.

References

[1] M. ABADI, L. LAMPORT AND P. WOLPER, *Realizable and unrealizable specifications of reactive systems*, in: Proc. 16th Int. Coll. Automata, Languages and Programming, LNCS **372** (1989) 1–17.

[2] R. B. BOPPANA AND M. SIPSER, *The complexity of finite functions*, in: J. VAN LEEUWEN Ed., Handbook of Theoretical Computer Science, Volume A, Elsevier, 1990, 757–804.

[3] J. BRADFIELD, J. ESPARZA AND A. MADER, *An effective tableau system for the linear time μ-calculus*, in: Proc. 23rd Int. Coll. Automata, Languages and Programming, LNCS **1099** (1996) 98–109.

[4] J. R. BÜCHI AND L. H. LANDWEBER, *Solving sequential conditions by finite state strategies*, Trans. Amer. Math. Soc. **138** (1969) 295–311.

[5] O. BURKART AND B. STEFFEN, *Model checking for context-free processes*, in: Proc. 4th Int. Conf. on Computer Aided Verification, LNCS **630** (1992) 123–137.

[6] M. DAM, *CTL* and ECTL* as a fragments of the modal μ-calculus*, in: CAAP'92, LNCS **581** (1992) 145–165.

[7] E. A. EMERSON, *Automata, tableaux, and temporal logics (Extended abstract)*, in: Logics of Programs, LNCS **193** (1985) 79–88.

[8] E. A. EMERSON AND C. S. JUTLA, *The complexity of tree automata and logics of programs*, in: Proc. of 29th Annual IEEE Symp. on Foundations of Computer Science (1988) 328-337.

[9] E. A. EMERSON AND C. S. JUTLA, *Tree automata, mu-calculus and determinacy*, in: Proc. of 32nd Annual IEEE Symp. on Foundations of Computer Science (1991) 368-377.

[10] E. A. EMERSON, C. S. JUTLA AND A. P. SISTLA, *On model-checking for fragments of μ-calculus*, in: Proc. 5th Int. Conf. on Computer Aided Verification, LNCS **697** (1993) 383–396.

[11] Y. GUREVICH AND L. HARRINGTON, *Trees, automata, and games*, in: Proc. of 14th Annual ACM Symp. on Theory of Computing (1982) 60–65.

[12] H. LESCOW, *On polynomial-size programs winning finite-state games*, in: Proc. 7th Int. Conf. on Computer Aided Verification, LNCS **939** (1995) 239–252.

[13] N. KLARLUND, *Progress measures, immediate determinacy, and a subset construction for tree automata*, in: Proc. of 7th Annual IEEE Symp. on Logic in Computer Science (1992) 382-393.

[14] O. MALER, A. PNUELI AND J. SIFAKIS *On the synthesis of discrete controllers for timed systems*, in: Proc. of 12th Annual Symp. on Theoretical Aspects of Computer Science, LNCS **900** (1995) 229–242.

[15] R. MCNAUGHTON, *Infinite games played on finite graphs*, Ann. Pure Appl. Logic **65** (1993) 149–184.

[16] A. NERODE, J. B. REMMEL AND A. YAKHNIS, *McNaughton games and extracting strategies for concurrent programs*, Ann. Pure Appl. Logic **78** (1996) 203–242.

[17] A. PNUELI AND R. ROSNER, *On the synthesis of a reactive module*, in: Proc. 16th Annual ACM Symp. on Principles of Programming Languages (1989) 179–190.

[18] C. STIRLING, *Modal and temporal logics for processes*, to appear in LNCS.

[19] W. THOMAS, *Automata on Infinite Objects*, in: J. VAN LEEUWEN Ed., Handbook of Theoretical Computer Science Volume B, Elsevier, 1990, 133–192.

[20] W. THOMAS, *On the synthesis of strategies in infinite games*, in: Proc. of 12th Annual Symp. on Theoretical Aspects of Computer Science, LNCS **900** (1995) 1–13.

[21] I. WALUKIEWICZ, *Completeness of Kozen's axiomatisation of the propositional mu-calculus*, in: Proc. of 10th Annual IEEE Symp. on Logic in Computer Science (1995) 14–24.

[22] I. WALUKIEWICZ, *Pushdown processes: games and model checking*, in: Proc. 8th Int. Conf. on Computer Aided Verification, LNCS **1102** (1996) 62–74.

[23] W. ZIELONKA, *Infinite games on finitely coloured graphs with applications to automata on infinite trees*, Technical Report, Universitè Bordeaux I, 1994, to appear in TCS.