# Playing Games on Automata

Orna Kupferman

School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel

**Abstract.** The interaction between a system and its environment corresponds to a game in which the system and the environment generate a word over the alphabet of assignments to the input and output signals. The system player wins if the generated word satisfies a desired specification. A correct reactive system has to satisfy its specification in all environments, and thus corresponds to a winning strategy for the system player in the above game. This view is the key to the game-based approach for reactive synthesis. It reduces synthesis to the problem of generating winning strategies in two-player games whose winning conditions are on-going behaviors, which can be specified by automata on infinite words. The paper presents the game-based approach, focusing on ways to cope with the fact that the game cannot be played over nondeterministic automata.

## 1 Introduction

*Synthesis* is the automated construction of systems from their specifications [8,19]. Synthesis enables designers to focus on *what* the system should do rather on *how* it should do it. A reactive system interacts with its environment and should *realize* the specification, namely satisfy it against all possible environments. More formally, the specification is a language $L$ of infinite words over the alphabet $2^{I \cup O}$, where $I$ and $O$ are sets of input and output signals, respectively, and the goal is to construct a reactive system that outputs assignments to the signals in $O$ upon receiving assignments to the signals in $I$, such that the generated sequence of assignments, which can be viewed as an infinite computation in $(2^{I \cup O})^\omega$, is in $L$ [19].

The common approach for solving the synthesis problem is to define a two-player game on top of a deterministic automaton $\mathcal{D}$ for $L$. The positions of the game are the states of $\mathcal{D}$. In each round of the game, one player (the environment) provides an assignment in $2^I$ to the input signals, the second player (the system) responds with an assignment in $2^O$ to the output signals, and the game proceeds to the corresponding successor state. More formally, when the game is in a position that corresponds to a state $q$ of $\mathcal{D}$, the environment provides $i \in 2^I$, the system responds with $o \in 2^O$, and the game moves to the position associated with the single $(i \cup o)$-successor of $q$. The goal of the system is to respond in a way so that the sequence of visited positions satisfies the acceptance condition of $\mathcal{D}$, indicating that the generated computation is in $L$. The system has a winning strategy in the game iff the language $L$ is realizable [11]. Moreover, an

$(I/O)$-*transducer* that models a system that realizes the specification can be constructed form the winning strategy.

Now, if one replaces $\mathcal{D}$ by a nondeterministic automaton $\mathcal{A}$ for $L$, the response of the system to an assignment $i \in 2^I$ should include, in addition to an assignment $o \in 2^O$, also an $(i \cup o)$-successor of of the current position. Letting the system resolve the nondeterminism makes sense, in particular that $\mathcal{A}$ has an accepting run on all computations in $L$. Letting the system resolve the nondeterminism is, however, problematic. The problem is that the choice of an $(i \cup o)$-successor should accommodate all possible future choices of the environment. In particular, if different future choices of the environment induce computations that are all in the language of $\mathcal{A}$ yet require different nondeterministic choices, the system cannot win. Thus, it might be that $L$ is realizable and still the system has no winning strategy in the game played on $\mathcal{A}$.

The need to play the synthesis game on a deterministic automaton is a real barrier in practice. First, determinization constructions are very complicated and their implementation is very difficult. Second, they require the use of winning conditions that are more complex. In this paper we explain further the difficulty that nondeterminism imposes on games and survey two approaches that have been suggested to cope with it.

The first approach, introduced on [15], is based on *determinancy of games*, and the fact that the use of nondeterministic automata involves a *one-sided error*. To see the idea, consider a specification $L$, and recall that playing a game on a nondeterministic automaton for $L$ may result in the system player losing even when $L$ is realizable. Determinancy of games implies that whenever $L$ is not realizable by the system, its complement is realizable by the environment. Thus, realizability of $L$ can be checked also by playing the synthesis game on an automaton for its complement. In Section 4 we explain how the one-sided errors in the two checks can be combined in an algorithm that is played on automata for both $L$ and its complement, and describe two interesting heuristics for reducing the one-sided error and coping with. The first heuristic refines the game by checking the *universal satisfiability* of the specification and its complement, and the second heuristic performs a subset-construction (rather than full determinization) on the underlying automata.

The second approach is based on playing the game on nondeterministic automata that can resolve their nondeterministic choices in a way that only depends on the past. The study of such automata started in [16], by means of tree automata for derived languages. It then continued by means of *good for games* word automata [12]. The idea is also used in [9] in the framework of cost functions, and we adopt the term *history determinism* that is used there.

Formally, a nondeterministic automaton $\mathcal{A}$ over an alphabet $\Sigma$ and with state space $Q$ is history deterministic (HD, for short) if there is a *strategy* $g : \Sigma^* \to Q$, such that for every word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$, the sequence $g(w) = g(\epsilon)$, $g(\sigma_1)$, $g(\sigma_1 \cdot \sigma_2), \ldots$ is a run of $\mathcal{A}$ on $w$, and whenever $w$ is accepted by $\mathcal{A}$, the run $g(w)$ is accepting. Thus, the strategy $g$ maps each word $x \in \Sigma^*$ to the state that is visited after $x$ is read.

In Section 5 we discuss HD automata, explain their application in the solution for the synthesis problem, and survey main results about them. We focus on the difference between HD automata and deterministic ones, and show that not only HD automata cannot always be pruned to deterministic ones [5], HD co-Büchi automata may in fact be exponentially more succinct than deterministic ones [13]. We also discuss the fact that for synthesis of $(I, O)$-transducers, one can work with $(I/O)$-*aware HD automata*, whose history-determinism can be defined with respect to the partition of the involved signals to inputs and outputs [10].

## 2 Preliminaries

### 2.1 Automata

A *nondeterministic word automaton* over a finite alphabet $\Sigma$ is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where $Q$ is a set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to 2^Q \setminus \{\emptyset\}$ is a total transition function, and $\alpha$ is an acceptance condition. We say that $\mathcal{A}$ is *deterministic* if for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q, \sigma)| = 1$. For a state $q \in Q$, we use $\mathcal{A}^q$ to denote the automaton $\mathcal{A}$ with initial state $q$. That is, $\mathcal{A}^q = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$. We sometimes refer to the transition function $\delta$ as a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, where for every two states $q, s \in Q$ and letter $\sigma \in \Sigma$, we have that $\langle q, \sigma, s \rangle \in \Delta$ iff $s \in \delta(q, \sigma)$.

A *run* of $\mathcal{A}$ on $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is an infinite sequence of transitions $r = \langle r_0, \sigma_1, r_1 \rangle, \langle r_1, \sigma_2, r_2 \rangle, \langle r_2, \sigma_3, r_3 \rangle, \ldots \in \Delta^\omega$, such that $r_0 = q_0$, and for all $i \geq 0$, we have that $r_{i+1} \in \delta(r_i, \sigma_{i+1})$. The acceptance condition $\alpha$ determines which runs are "good". We consider here the *Büchi* and *co-Büchi* transition-based acceptance conditions, where $\alpha \subseteq \Delta$ is a subset of transitions. For a run $r$, let $inf(r) \subseteq \Delta$ be the set of transitions that $r$ traverses infinitely often. Thus, $inf(r) = \{t \in \Delta : t = \langle r_i, \sigma_{i+1}, r_{i+1} \rangle \text{ for infinitely many } i\text{'s}\}$. A run $r$ of a Büchi automaton is *accepting* iff it traverses transitions in $\alpha$ infinitely often, thus $inf(r) \cap \alpha \neq \emptyset$. Dually, a run $r$ of a co-Büchi automaton is accepting iff it traverses transitions in $\alpha$ only finitely often, thus $inf(r) \cap \alpha = \emptyset$. A run that is not accepting is *rejecting*. Note that as $\mathcal{A}$ is nondeterministic, it may have several runs on a word $w$. A word $w$ is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on $w$. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. Two automata are *equivalent* if their languages are equivalent.

We use three letter acronyms in $\{\text{D,N}\} \times \{\text{B,C}\} \times \{\text{W}\}$ to denote classes of word automata. The first letter indicates whether this is a deterministic or nondeterministic automaton, and the second indicates the acceptance condition. For example, NBW is a nondeterministic Büchi automaton.

We say that a nondeterministic automaton is *history deterministic* (HD, for short) if its nondeterminism can be resolved based on the past [12]. Formally, a nondeterministic automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ is HD if there exists a function $g : \Sigma^* \to Q$ such that the following hold.

- The strategy $g$ is compatible with $q_0$ and $\delta$. That is, $g(\epsilon) = q_0$, and for every $w \in \Sigma^*$ and $\sigma \in \Sigma$, we have that $\langle g(w), \sigma, g(w \cdot \sigma) \rangle \in \Delta$.

– The strategy $g$ "covers" all words in $L(\mathcal{A})$. That is, for every word $w = \sigma_1 \cdot \sigma_2 \cdots \in L(\mathcal{A})$, the run that $g$ induces on $w$, namely $\langle g(\epsilon), \sigma_1, g(\sigma_1)\rangle$, $\langle g(\sigma_1), \sigma_2, g(\sigma_1 \cdot \sigma_2)\rangle, \ldots$, is accepting.

We then say that $g$ *witnesses* the HDness of $\mathcal{A}$. Note that a state $q$ of $\mathcal{A}$ may be reachable via different words, and $g$ may suggest different transitions from $q$ after different words are read. Thus, $g$ need not induce a prunning of $\mathcal{A}$ to a deterministic automaton. Still, $g$ itself is a deterministic strategy that resolves the nondeterminism in $\mathcal{A}$ in a way that only depends only on the past, namely on the word read so far.

### 2.2 Games

We consider *two-player turn-based* games played on graphs. The vertices of the game graph are partitioned between the players. Starting from an initial vertex, the players jointly move a token along the graph, with each player deciding the successor vertex whenever the token reaches a vertex he owns. Formally, a *game* is a tuple $G = \langle V_{\text{AND}}, V_{\text{OR}}, E, \alpha\rangle$, where $V_{\text{AND}}$ and $V_{\text{OR}}$ are disjoint sets of positions, owned by Player AND and Player OR, respectively. Let $V = V_{\text{AND}} \cup V_{\text{OR}}$. Then, $E \subseteq V \times V$ is an edge relation, and $\alpha$ is a winning condition, defining a subset of $V^\omega$. A *play* is an infinite sequence of positions $v_0, v_1, \cdots \in V^\omega$, such that for every index $i \geq 0$, we have that $\langle v_i, v_{i+1}\rangle \in E$. A play $\pi \in V^\omega$ is *winning* for Player OR if $\pi$ satisfies $\alpha$, and is winning for Player AND otherwise. We focus here on *Büchi* games, where $\alpha \subseteq V$ and $\pi$ satisfies $\alpha$ if it visits the positions in $\alpha$ infinitely often. Note that since a play is winning for Player AND iff it is not winning for Player OR, the game is a co-Büchi game from the point of view of Player AND.

Recall that starting from some position $v_0 \in V$, the players generate a play in $G$ by letting the player that owns the current position to choose a successor. Formally, in every round of the game, if the current position is $v \in V_j$, for $j \in \{\text{AND}, \text{OR}\}$, then Player $j$ chooses a successor $v'$ of $v$, and the play proceeds to position $v'$. A *strategy* for a player $j \in \{\text{AND}, \text{OR}\}$ is a function $f_j : V^* \times V_j \to V$ such that for every $u \in V^*$ and $v \in V_j$, we have that $\langle v, f_j(u, v)\rangle \in E$. Thus, a strategy for Player $j$ maps the history of the game so far, when it ends in a position $v$ owned by Player $j$, to a successor of $v$. A strategy $f_j$ is *memoryless* if for every $u_1, u_2 \in V^*$ and $v \in V_j$, we have that $f_j(u_1 \cdot v) = f_j(u_2 \cdot v)$. Intuitively, a memoryless strategy for player $j$ maps all the histories of the game that end in a position $v$, owned by player $j$, to the same successor of $v$. Note that we can describe a memoryless strategy for Player $j$ by $f_j : V_j \to V$.

Two strategies $f_{\text{AND}}, f_{\text{OR}}$, and an initial position $v_0$ induce a play $\pi = v_0$, $v_1$, $v_2 \cdots \in V^\omega$, where for every $i \geq 0$, if $v_i \in V_j$, for $j \in \{\text{AND}, \text{OR}\}$, then $v_{i+1} = f_j((v_0, \ldots, v_{i-1}), v_i)$. We say that $\pi$ is the *outcome* of $f_{\text{OR}}$ and $f_{\text{AND}}$ from $v_0$, and denote $\pi = \mathit{outcome}(v_0, f_{\text{OR}}, f_{\text{AND}})$.

We say that a position $v \in V$ is winning for Player OR if there exists a strategy $f_{\text{OR}}$ such that for every strategy $f_{\text{AND}}$, we have that $\mathit{outcome}(v, f_{\text{OR}}, f_{\text{AND}})$

is winning for Player OR. We then say that $f_{\text{OR}}$ is a *winning strategy* of Player OR from $v$. We define similarly winning positions and strategies for Player AND.

It is known that Büchi games are *determined*. That is, every position in a Büchi game is winning for exactly one of the players. Solving a game amounts to deciding which vertices are winning for Player OR. Since Büchi games are determined, the other vertices are winning for Player AND. Büchi games can be solved in quadratic time [23]. Moreover, a player wins in a Büchi game iff he has a memoryless winning strategy [22].

### 2.3 Synthesis

Consider two finite sets $I$ and $O$ of input and output signals, respectively. For two words $w_I = i_0 \cdot i_1 \cdot i_2 \cdots \in (2^I)^\omega$ and $w_O = o_0 \cdot o_1 \cdot o_2 \cdots \in (2^O)^\omega$, we define $w_I \oplus w_O$ as the word in $(2^{I \cup O})^\omega$ obtained by merging $w_I$ and $w_O$. Thus, $w_I \oplus w_O = (i_0 \cup o_0) \cdot (i_1 \cup o_1) \cdot (i_2 \cup o_2) \cdots$.

An $(I/O)$-*transducer* models a finite-state system that generates assignments to the output signals while interacting with an environment that generate assignments to the input signals. Formally, an $(I/O)$-transducer is $\mathcal{T} = \langle I, O, S, s_0, \rho, \tau \rangle$, where $S$ is a set of states, $s_0 \in S$ is an initial state, $\rho : S \times 2^I \to S$ is a transition function, and $\tau : S \times 2^I \to 2^O$ is labelling function.

The transducer $\mathcal{T}$ models an interaction between an environment that generates at each moment in time a letter in $2^I$ and a system that responds with letters in $2^O$. Consider an input word $w_I = i_0 \cdot i_1 \cdots \in (2^I)^\omega$. The *run* of $\mathcal{T}$ on $w_I$ is the sequence $s_0, s_1, s_2 \ldots$ such that for all $j \geq 0$, we have that $s_{j+1} = \rho(s_j, i_j)$. The *output* of $\mathcal{T}$ on $w_I$ is then $w_O = o_1 \cdot o_2 \cdot \cdots \in (2^O)^\omega$, where $o_j = \tau(s_j, i_j)$ for all $j \geq 1$. Note that the environment initiates the interaction, with input $i_0$, to which the system responds with $\tau(s_0, i_0)$. The *computation of* $\mathcal{T}$ *on* $w_I$, denoted $\mathcal{T}(w_I)$, is then $w_I \oplus w_O$. Thus, $\mathcal{T}(w_I) = i_0 \cup o_1, i_1 \cup o_2, \ldots \in (2^{I \cup O})^\omega$.

Consider a specification $L \subseteq (2^{I \cup O})^\omega$. We say that an $(I/O)$-transducer $\mathcal{T}$ $(I/O)$-*realizes* $L$ if for every input word $w_I \in (2^I)^\omega$, the computation of $\mathcal{T}$ on $w_I$ is in $L$. In the *synthesis* problem, we are given a specification $L \subseteq (2^{I \cup O})^\omega$ and we have to decide whether $L$ is $(I/O)$-realizable. In case the answer is positive, we also need to return an $(I/O)$-transducer that $(I/O)$-realizes $L$. The language $L$ is typically given by a formula $\psi$ in *linear temporal logic* (LTL, for short) [18] Formulas of LTL are constructed from the set $I \cup O$ of signals using the usual Boolean operators and the temporal operators $G$ ("always") and $F$ ("eventually"), $X$ ("next time") and $U$ ("until"). The language induced by $\psi$ is then $L_\psi = \{w : w \models \psi\}$.

## 3 From Synthesis to Games on Automata

In this section we describe the game-based approach to synthesis and explain why it is traditionally applied with deterministic automata. The game-based approach first translates the specification to an automaton:

**Theorem 1.** [24] *For every LTL formula $\psi$, there is an NBW $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states such that $L(\mathcal{A}_\psi) = \{w : w \models \psi\}$.*

The input to the synthesis problems consists of the specification $\psi$, which we translate to an NBW, and a partition of the signals in $\psi$ to $I$ and $O$. Given an NBW $\mathcal{A} = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$ for the specification we wish to synthesize, we define the *synthesis game* $G_{\text{SYN}}(\mathcal{A}, I, O)$ as follows. The game proceeds in rounds. Let $q$ be the position of the game at the beginning of some round. The round proceeds as follows. First, Player AND (the environment) chooses a letter $i \in 2^I$. Then, Player OR (the system) chooses a letter $o \in 2^O$ and a state $q' \in \delta(q, i \cup o)$, and the game proceeds to the position $q'$. Formally, we define $G_{\text{SYN}}(\mathcal{A}, I, O) = \langle V_{\text{AND}}, V_{\text{OR}}, E, \alpha \rangle$, where

- $V_{\text{AND}} = Q$,
- $V_{\text{OR}} = Q \times 2^I$, and
- $E = \{\langle q, \langle q, i \rangle\rangle : q \in V_{\text{AND}} \text{ and } i \in 2^I\} \cup \{\langle \langle q, i \rangle, q' \rangle : \text{there is } o \in 2^O \text{ such that } q' \in \delta(q, i \cup o)\}$.

Intuitively, Player AND provides a sequence of assignments to the input signals, and Player OR responds with a sequence of assignments to the output signals and a run of $\mathcal{A}$ on the generated computation, aiming for a response that is an accepting run. We say that an automaton $\mathcal{A}$ over alphabet $2^{I \cup O}$ is *sound for $(I/O)$-realizability* if $q_0$ being a winning state for Player OR in $G_{\text{SYN}}(\mathcal{A}, I, O)$ implies that $\mathcal{A}$ is $(I/O)$-realizable. Then, $\mathcal{A}$ is *complete for $(I/O)$-realizability* if $(I/O)$-realizability of $\mathcal{A}$ implies that $q_0$ is a winning state for Player OR in $G_{\text{SYN}}(\mathcal{A}, I, O)$.

*Example 1.* Let $I = \{i\}$, $O = \{o\}$, and consider the specification $\psi = (i \leftrightarrow o) \wedge Xo$. Clearly, $\psi$ is $(I/O)$-realizable, for example by a transducer that copies to $o$ the value of $i$ in the first cycle of the interaction and assigns true to $o$ in the second cycle. In Figure 1, we describe the DBW $\mathcal{D}_\psi$, which accepts exactly all the computations that satisfy $\psi$, and the game $G_{\text{SYN}}(\mathcal{D}_\psi, I, O)$. We label the transitions of $\mathcal{D}_\psi$ by strings in $\{00, 01, 10, 11\}$, indicating the assignment to $i$ (the first bit) and $o$ (the second bit). We label the edges of $G_{\text{SYN}}(\mathcal{D}_\psi, I, O)$ by 0 or 1, indicating the assignment to $i$ (in edges that leave vertices own by Player AND – the environment) and $o$ (in edges that leave vertices owned by Player OR – the system). Since the only cycles in $\mathcal{D}_\psi$ are the accepting and rejecting sinks, we view $G_{\text{SYN}}(\mathcal{D}_\psi, I, O)$ as a reachability game with an objective $q_{acc}$.

It is easy to see that Player OR wins, with a strategy that moves the token to $q_{acc}$, namely one that copies to $o$ the value of $i$ in the first round and assigns true to $o$ in the second round. □

We prove that all deterministic automata are sound and complete for $(I/O)$-realizability, all nondeterministic ones are sound, yet nondeterministic automata may not be complete.

**Theorem 2.** *Nondeterministic automata are sound for $(I/O)$-realizability.*

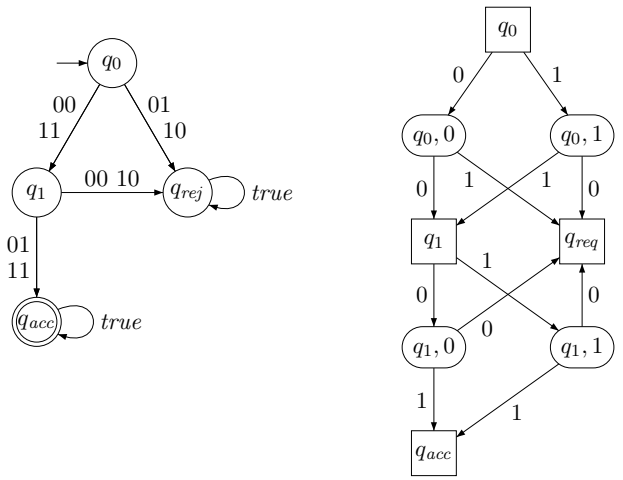


Fig. 1: The DBW $\mathcal{D}_\psi$ (left) and the game $G_{\text{SYN}}(\mathcal{D}_\psi, I, O)$ (right).

*Proof.* Consider a nondeterministic automaton $\mathcal{A} = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$. We show that if $q_0$ is winning for Player OR in $G_{\text{SYN}}(\mathcal{A}, I, O)$, then $\mathcal{A}$ is $(I/O)$-realizable.

Recall that the winner in a Büchi game has a memoryless winning strategy. Assume that $q_0$ is winning for Player OR in $G_{\text{SYN}}(\mathcal{A}, I, O)$, and Let $f_{\text{OR}} : Q \times 2^I \to Q$ be a memoryless winning strategy for him. Recall that $f_{\text{OR}}(\langle q, i \rangle) = q'$ implies that there is $o \in 2^O$ such that $q' \in \delta(q, i \cup o)$. Thus, we can view the strategy of Player OR as a function $f_{\text{OR}} : Q \times 2^I \to Q \times 2^O$, which also specifies the assignment to the output signals according to which the successor state is chosen. Then, the strategy $f_{\text{OR}}$ induces the $(I/O)$-transducer $\mathcal{T} = \langle I, O, Q \times 2^O, \langle q_0, \emptyset \rangle, \rho, \tau \rangle$, where $\rho$ and $\tau$ are induced from $f_{\text{OR}}$ in the expected way, thus for all states $q \in Q$ and inputs $i \in 2^I$, we have that $\langle \rho(q, i), \tau(q, i) \rangle = f_{\text{OR}}(q, i)$. □

**Theorem 3.** *Deterministic automata are complete for $(I/O)$-realizability.*

*Proof.* Consider a deterministic automaton $\mathcal{A} = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$. We show that if $\mathcal{A}$ is $(I/O)$-realizable, then $q_0$ is winning for Player OR in $G_{\text{SYN}}(\mathcal{A}, I, O)$.

Assume that $\mathcal{A}$ is $(I/O)$-realizable. Let $\mathcal{T} = \langle I, O, S, s_0, \rho, \tau \rangle$ be an $(I/O)$-transducer that $(I/O)$-realizes $\mathcal{A}$, and let $G_{\text{SYN}}(\mathcal{A}, I, O) = \langle Q, Q \times 2^I, E, \alpha \rangle$. We describe a (not necessarily memoryless) winning strategy $f_{\text{OR}}$ for Player OR in $G_{\text{SYN}}(\mathcal{A}, I, O)$. For every prefix of a play $\pi = q_0, \langle q_0, i_0 \rangle, q_1, \langle q_1, i_1 \rangle, \ldots, q_k, \langle q_k, i_k \rangle$, we define $f_{\text{OR}}(\pi) = \delta(q_k, i_k \cup \tau(\rho(s_0, i_0 \cdot i_1 \cdots i_{k-1}), i_k)$. That is, $f_{\text{OR}}$ follows the run of $\mathcal{T}$ on the generated computation, and when $\mathcal{T}$ is in state $s$ and Player AND provides an assignment $i \in 2^I$, it extends the computation according to $\tau(s, i)$ and extends the run by moving to $\rho(s, i)$.

By the definition of $G_{\text{SYN}}(\mathcal{A}, I, O)$, all the outcomes of the game when Player OR uses $f_{\text{OR}}$ are computations of $\mathcal{T}$. Since $\mathcal{T}$ $(I/O)$-realizes $\mathcal{A}$, the run of $\mathcal{A}$ on them is accepting, and so $f_{\text{OR}}$ is indeed winning for Player OR. □

**Theorem 4.** *Nondeterministic automata are not complete for $(I/O)$-realizability.*

*Proof.* Let $I = \{i\}$, $O = \{o\}$, and recall the specification $\psi = (i \leftrightarrow o) \wedge Xo$ discussed in Example 1. In the example, we described a DBW $\mathcal{D}_\psi$ for $\psi$, inducing a game in which Player OR wins. In Figure 2, we describe an NBW $\mathcal{A}_\psi$ for $\psi$. The nondeterminism in $\mathcal{A}_\psi$ is artificial, and corresponds to guessing whether the assignment to $i$ in the second cycle of the interaction is false (in which case an accepting run is via the state $q_1$), or true (in which case an accepting run is via the state $q_2$).

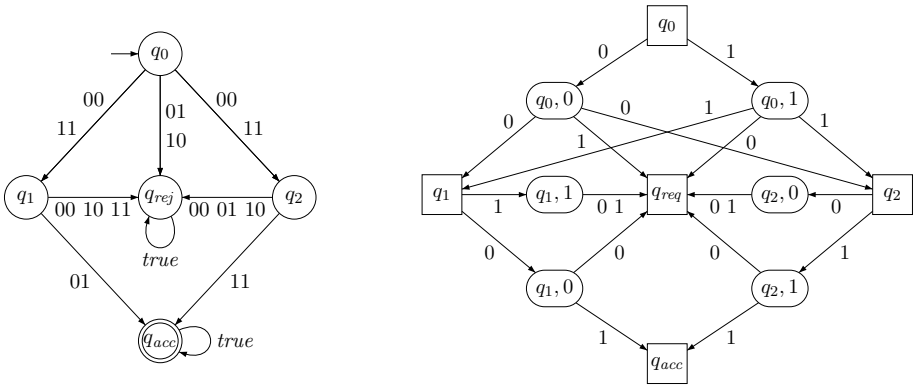


Fig. 2: The NBW $\mathcal{A}_\psi$ (left) and the game $G_{\text{SYN}}(\mathcal{A}_\psi, I, O)$ (right).

The game $G_{\text{SYN}}(\mathcal{A}_\psi, I, O)$ appears on the right. Note that if Player OR moves the token to $q_1$, Player AND responds with 0 (that is, $\neg i$), and if Player OR moves the token to $q_2$, Player AND responds with 1 (that is, $i$). In both cases, regardless of the response of Player OR in the next round, the token reaches $q_{rej}$. Thus, while $\mathcal{A}_\psi$ is $(I/O)$-realizable, Player OR does not have a winning strategy from $q_0$ in $G_{\text{SYN}}(\mathcal{A}_\psi, I, O)$. □

Beyond getting convinced about the specific example in the proof of Theorem 4, it is important to understand the conceptual difficulty that nondeterminism imposes on games. Whenever a nondeterministic automaton takes a nondeterministic transition, it makes a guess about the future. For example, when the NBW $\mathcal{A}_\psi$ from the proof branches from $q_0$ to $q_1$, it guesses that the next assignment to $i$ is false, and when it branches to $q_2$, it guesses that the next assignment to $i$ is true. While $\mathcal{A}_\psi$ has an accepting run on all the words in $L(\mathcal{A}_\psi)$, forcing Player OR to choose a successor state forces $\mathcal{A}_\psi$ to accept all the words that share the prefix generated so far with that same nondeterministic choice. In the

example, when Player OR moves the token from $\langle q_0, 0\rangle$ or $\langle q_0, 1\rangle$ to the vertex $q_1$ in $G_{\text{SYN}}(\mathcal{A}_\psi, I, O)$, it restricts the runs of $\mathcal{A}_\psi$ on the generated words to ones that move from $q_0$ to $q_1$, namely to ones that guess that the next assignment to $i$ is false, implying that Player AND wins by assigning true to $i$.

## 4 Using Determinancy of Games

The roles of the system and the environment in the synthesis setting may be switched, and we can talk about a formula $\psi$ being $(O/I)$-realizable by the environment, meaning that there is an $(O/I)$-transducer $\mathcal{T}$ such that for every output sequence $w_O = o_0, o_1, o_2, \ldots \in (2^O)^\omega$, the computation of $\mathcal{T}$ on $w_O$ satisfies $\psi$. Note that in order for switched settings to complement each other, we let the environment move first in both types of realizability. Thus, the settings are not completely dual. From determinancy of games, we know that for every specification $\psi$, either $\psi$ is $(I/O)$-realizable by the system or $\neg\psi$ is $(O/I)$-realizable by the environment.

The soundness of nondeterministic automata for realizability suggests the following simple heuristic: Given an LTL formula $\psi$ over $I \cup O$, let $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$ be NBWs for $\psi$ and $\neg\psi$, respectively. We proceed as follows.

**(1)** Solve the game $G_{\text{SYN}}(\mathcal{A}_\psi, I, O)$.
  **(1.1)** If the system wins, we are done: $\psi$ is $(I/O)$-realizable and the winning strategy for the system induces a transducer that $(I/O)$-realizes $\psi$.
  **(1.2)** If the environment wins, proceed to (2).
**(2)** Solve the game $G_{\text{SYN}}(\mathcal{A}_{\neg\psi}, O, I)$.
  **(2.1)** If environment wins, we are done. Indeed, if $\neg\psi$ is $(O/I)$-realizable, then $\psi$ is not $(I/O)$-realizable.
  **(2.2)** If the system wins, proceed to (3).
**(3)** Continue to read this section.

Since $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$ are sound but not complete, we may reach Step 3 in the algorithm. Below we discuss a heuristic for coping with this situation.

Consider an LTL formula $\psi$ over $I \cup O$. We say that $\psi$ is *universally* $(I/O)$-*satisfiable* if for every input sequence $w_I \in (2^I)^\omega$, there is an output sequence $w_O \in (2^O)^\omega$ such that $w_I \oplus w_O$ satisfies $\psi$. It is not hard to see that $(I/O)$-realizability implies universal $(I/O)$-satisfiability, but not the other way around. For example, letting $I = \{i\}$ and $O = \{o\}$, the formula $\psi = G(Xi \leftrightarrow o)$ is universally $(I/O)$-satisfiable but not $(I/O)$-realizable. Indeed, once a transducer $\mathcal{T}$ sets the value of the signal $o$ in the first round, the environment can assign $\neg o$ to $i$ in the next round, causing the generated computation to violate $Xi \leftrightarrow o$ and hence also violate $\psi$.

As is the case with realizability, the roles of the system and the environment may be switched also in the context of universal satisfiability, and we can talk about a formula $\psi$ being universally $(O/I)$-satisfiable, meaning that for every output sequence $w_O \in (2^O)^\omega$, there is an input sequence $w_I \in (2^I)^\omega$ such that $w_I \oplus w_O$ satisfies $\psi$. Note that for universal satisfiability, the identity of the player

that moves first is irrelevant, and the definitions of universal $(I/O)$-satisfiability and universal $(O/I)$-satisfiability are completely dual.

Checking universal satisfaction is much simpler than checking realizability, not just from a theoretical point of view (the problem is EXPSPACE-complete [21]), but also in practice – universal satisfaction amounts to checking universality of an NBW, which can be done without determinization [17].

Below we describe the methodology for using universal satisfiability in the process of checking realizability.

Given an LTL formula $\psi$ over $I$ and $O$, we proceed as follows.

**(1)** Check universal $(I/O)$-satisfiability of $\psi$.
- **(1.1)** If the answer is negative, we are done. Indeed, if $\psi$ is not universally $(I/O)$-satisfiable, then $\psi$ is also not $(I/O)$-realizable.
- **(1.2)** If the answer is positive, proceed to (2).

**(2)** Check universal $(O/I)$-satisfiability of $\neg\psi$.
- **(2.1)** If the answer is negative, we are done. Indeed, if $\neg\psi$ is not universally $(O/I)$-satisfiable, then $\neg\psi$ is not $(O/I)$-realizable by the environment, implying that $\psi$ is $(I/O)$-realizable by the system. Moreover, a transducer for $\psi$ can simply generate the output sequence $w_O \in (2^O)^\omega$ for which for all $w_I \in (2^I)^\omega$ we have that $w_I \oplus w_O \models \psi$. The existence of such a sequence follows from $\neg\psi$ not being universally $(O/I)$-satisfiable, and the algorithm for checking universal $(O/I)$-satisfiability returns it.
- **(2.2)** If the answer is positive, proceed to (3).

**(3)** This is the interesting case: both $\psi$ is universally $(I/O)$-satisfiable and $\neg\psi$ is universally $(O/I)$-satisfiable. Note that while it cannot be that both $\psi$ and $\neg\psi$ are realizable, they can both be universally satisfiable. When this happens, we know that one of the players, the system or the environment, cannot arrange the responses that work for the universal satisfiability in the form of the strategy that is needed for realizability. In this case, continue to read this section.

As an example to an LTL formula that reaches Step (3) consider $\psi = G(Xi \leftrightarrow o)$. As discussed above, the formula $\psi$ is universally $(I/O)$-satisfiable but not $(I/O)$-realizable. Note that $\neg\psi = F\neg(Xi \leftrightarrow o)$, which is universally $(O/I)$-satisfiable, and would thus reach Step (3). In fact, all LTL formulas $\psi$ that are universally $(I/O)$-satisfiable but not $(I/O)$-realizable are such that $\neg\psi$ is $(O/I)$-realizable, and hence $\neg\psi$ is universally $(O/I)$-satisfiable, and so the application of the algorithm on them reaches Step (3).

Below we describe a heuristic for handling specifications that reach Step (3). The idea is to apply the subset construction on $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$, aiming to refine the universal satisfiability checks by taking into account the history of the game. In more detail, rather than checking the universal satisfiability of $\psi$ and $\neg\psi$ once, here we check the specifications induced by $\psi$ and $\neg\psi$ after each finite history of the interaction between the system and the environment. This is done by performing a subset construction on $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$. While the latter involves an exponential blow up in the state spaces of $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$, it is independent of the acceptance condition, and can be implemented symbolically.

Let $\mathcal{A}_\psi = \langle 2^{I \cup O}, S, S_0, \rho, \alpha \rangle$ and $\mathcal{A}_{\neg\psi} = \langle 2^{I \cup O}, S', S'_0, \rho', \alpha' \rangle$ be NBWs for $\psi$ and $\neg\psi$, respectively. Let $\mathcal{U}_\psi$ be the pre-automaton obtained by applying the subset construction to $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$. Thus, $\mathcal{U}_\psi = \langle 2^{I \cup O}, 2^S \times 2^{S'}, \langle S_0, S'_0 \rangle, \delta \rangle$, where for all $\langle P, P' \rangle \in 2^S \times 2^{S'}$ and $\sigma \in 2^{I \cup O}$, we have that $\delta(\langle P, P' \rangle, \sigma) = \langle \rho(P, \sigma), \rho'(P', \sigma) \rangle$. For a state $\langle P, P' \rangle$ of $\mathcal{U}_\psi$, let $L(\mathcal{A}^P_\psi)$ and $L(\mathcal{A}^P_{\neg\psi})$ be the languages of $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$ with initial sets $P$ and $P'$, respectively. We say that a set $P \in 2^S$ is *system hopeful* if for all $w_I \in (2^I)^\omega$, there is $w_O \in (2^O)^\omega$ such that $w_I \oplus w_O \in L(\mathcal{A}^P_\psi)$. We say that a set $P' \in 2^{S'}$ is *environment hopeful* if for all $w_O \in (2^O)^\omega$, there is $w_I \in (2^I)^\omega$ such that $w_I \oplus w_O \in L(\mathcal{A}^{P'}_{\neg\psi})$. Thus, system hopefulness coincides with universal $(I/O)$-satisfaction, except that instead of talking about satisfaction of an LTL formula we talk about the membership in the language of $\mathcal{A}^P_\psi$. Dually, environment hopefulness refers to membership in $\mathcal{A}^{P'}_{\neg\psi}$.

Consider a state $\langle P, P' \rangle \in 2^S \times 2^{S'}$ of $\mathcal{U}_\psi$. It is possible to decide in space exponential in the length of $\psi$ whether $P$ is system hopeful and whether $P'$ is environment hopeful. Indeed, the check is similar to the check for universal satisfaction. For the case of system hopefulness, we project $\mathcal{A}^P_\psi$ on $2^I$ and check that the obtained NBW is universal. For environment hopefulness we do the same, with $\mathcal{A}^{P'}_{\neg\psi}$ and a projection on $2^O$.

We can now describe the heuristic that can be applied in Step (3).

**(3)** Consider the game $G_{\text{SYN}}(\mathcal{U}_\psi, I, O)$.

**(3.1)** If the system has a strategy to reach a state $\langle P, P' \rangle$ such that $P$ is system-hopeful and $P'$ is not environment-hopeful, then we are done. Indeed, $\psi$ is realizable, and we can also have a transducer for it.

**(3.2)** If the environment has a strategy to reach a state $\langle P, P' \rangle$ such that $P'$ is environment-hopeful and $P$ is not system-hopeful, then we are done. Indeed, in a manner dual to the one above, $\neg\psi$ is realizable by the environment.

**(3.3)** If we get here, both the system and the environment have strategies to stay forever in the region of states that are both system-hopeful and environment-hopeful. At this point we give up and turn to solve the realizability problem using one of the traditional algorithms (or continue and read Section 5). The information gathered during our algorithm is still useful and enables us to restrict the realizability game to states in the region of hopeful states (all the other states are replaced by two states – one is winning for the system and one is winning for the environment).

## 5 Using History-Deterministic Automata

Recall that an HD automaton can resolve its nondeterministic choices in a way that only depends on the past. Obviously, there exist HD automata: deterministic ones, or nondeterministic ones that are *determinizable by pruning* (DBP); that is, ones that just add transitions on top of a deterministic automaton. In

fact, the HD automata constructed in [12] are DBP.[1] Before we describe some theoretical properties of HD automata, let us first show that the ability to resolve nondeterministic choices in a way that depends on the past makes HD automata sound and complete for synthesis.

**Theorem 5.** *History deterministic automata are sound and complete for $(I/O)$-realizability.*

*Proof.* Since all nondeterministic automata are sound for $(I/O)$-realizability, we only have to prove completeness. Consider an HD automaton $\mathcal{A} = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$. Let $g : (2^{I \cup O})^* \to Q$ be a strategy that witnesses the HDness of $\mathcal{A}$. We show that if $\mathcal{A}$ is $(I/O)$-realizable, then $q_0$ is winning for Player OR in $G_{\text{SYN}}(\mathcal{A}, I, O)$.

Assume that $\mathcal{A}$ is $(I/O)$-realizable. Let $\mathcal{T} = \langle I, O, S, s_0, \rho, \tau \rangle$ be an $(I/O)$-transducer that $(I/O)$-realizes $\mathcal{A}$, and let $G_{\text{SYN}}(\mathcal{A}, I, O) = \langle Q, Q \times 2^I, E, \alpha \rangle$. We describe a winning strategy $f_{\text{OR}}$ for Player OR in $G_{\text{SYN}}(\mathcal{A}, I, O)$. For every prefix of a play $\pi = q_0, \langle q_0, i_0 \rangle, q_1, \langle q_1, i_1 \rangle, \ldots, q_k, \langle q_k, i_k \rangle$, let $s_0, s_1, \ldots, s_k$ be the run of $\mathcal{T}$ on $i_0, i_1, \ldots, i_k$, and let $o_0, o_1, \ldots, o_k$ be the sequence of outputs generated by $\mathcal{T}$. Thus, for all $j \geq 0$, we have that $o_j = \tau(s_j, i_j)$. We define $f_{\text{OR}}(\pi) = g(i_0 \cup o_o \cdot i_1 \cup o_1 \cdots i_k \cup o_k)$. Since $g$ is an HD strategy for $\mathcal{A}$, the strategy $f_{\text{OR}}$ is well defined, in the sense that it returns a state $q_{k+1} \in \delta(q_k, i_k \cup o_k)$. By the definition of $G_{\text{SYN}}(\mathcal{A}, I, O)$, all the outcomes of the game when Player OR uses $f_{\text{OR}}$ are computations of $\mathcal{T}$. Since $\mathcal{T}$ $(I/O)$-realizes $\mathcal{A}$, these computations are in the language of $\mathcal{A}$, and as $g$ is an HD-strategy for $\mathcal{A}$, the run that $g$ induces on them is accepting. Hence, $f_{\text{OR}}$ is winning for Player OR, and we are done. □

*Remark 1.* [**$(I/O)$-aware HD automata**] The definition of HD automata involves strategies that resolve nondeterminism by mapping each prefix in $(2^{I \cup O})^*$ to the transition that should be taken after reading the prefix. In the context of synthesis, it is useful to distinguish between nondeterminism due to $I$ and $O$. In order for an automaton to be complete for $(I/O)$-realizability, both types on nondeterminism should be resolved in a way that depends only on the past; but while nondeterminism in $I$ is hostile, and all $I$-futures should be accepted, nondeterminism in $O$ is cooperative, and a single $O$-future may be accepted.

In [10], we introduced and studied $(I/O)$-*aware HD automata*, which make use of this distinction. More formally, an automaton $\mathcal{A}$ over $2^{I \cup O}$ is $(I/O)$-aware HD if for every word $w_I \in (2^I)^\omega$, if $w_I$ is *hopeful*, namely it can be paired with a word $w_O \in (2^O)^\omega$ to a computation accepted by $\mathcal{A}$, then the pairing as well as the accepting run of $\mathcal{A}$ can be produced in an on-line manner, thus in a way that only depends on the past. It is shown in [10] that $(I/O)$-aware HD automata are sound and complete for $(I/O)$-realizability, and are unboundedly more succinct than deterministic and even HD automata. □

[1] As explained in [12], the fact that the HD automata constructed there are DBP does not contradict their usefulness in practice, as their transition relation is simpler than the one of the embodied deterministic automaton and it can be defined symbolically.

It is not hard to see that in the context of automata on finite words, every HD automaton is DBP. For automata on infinite words, the situation is much more involved and many questions concerning HD automata are still open. We describe here two key results that highlight the difference between deterministic and HD automata on infinite words. We show that not only HD automata need not be DBP, they may be exponentially more succinct than deterministic ones. We start with HD-NBWs, and show that a strategy that witnesses HDness may need to resolve the nondeterminism in a different manner in different visits to the same state of the NBW.

**Theorem 6.** *[5] There are HD-NBWs that are not determinizable by prunning.*

*Proof.* Consider the NBW $\mathcal{A}$ appearing in Figure 3. We prove that $\mathcal{A}$ is HD and is not DBP. Note that $\mathcal{A}$ gets stuck (and rejects) when it reads words that are not in $(a0 + a1)^\omega$. We claim that $L(\mathcal{A}) = L$, for

$$L = \{w \in (a0 + a1)^\omega : w \text{ has infinitely many infixes of the form } a0a0 \text{ or } a1a1\}.$$

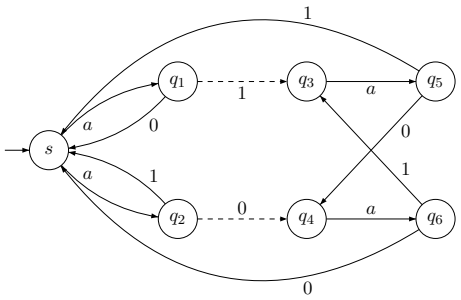


Fig. 3: A HD-NBW that is not DBP. Dashed transitions are in $\alpha$.

In order to see that $L(\mathcal{A}) \subseteq L$, note that if a word in $(a0 + a1)^\omega$ is not in $L$, and thus it has only finitely many infixes of the form $a0a0$ or $a1a1$, then it has a suffix $(a0a1)^\omega$. Also, when a run of $\mathcal{A}$ traverses an $\alpha$-transition when reading such a suffix, then after taking the $\alpha$-transition, it keeps looping at the $q_3, q_5, q_4, q_6$ cycle and never traverses an $\alpha$-transition again. In order to see that $L \subseteq L(\mathcal{A})$, note that after reading a prefix in $(a0 + a1)^*$, a run of $\mathcal{A}$ is in state $s$, $q_3$, or $q_4$. If the run is in state $s$ or $q_4$ and reads $a0a0$, then it can traverse an $\alpha$-transition and return to $s$, and if it is in state $q_3$ and reads $a0a0$, then it reaches the state $s$. Also, reading $a1$ from $s$, a run can return to $s$. Thus, reading $a0a0$ infinitely often enables a run to traverse $\alpha$-transitions infinitely often, and similarly for $a1a1$.

We continue and prove that $\mathcal{A}$ is HD. We do so by describing a strategy $f$ that witnesses its HDness. Note that there is one nondeterministic transition

in $\mathcal{A}$: reading $a$ in state $s$, a run can proceed to $q_1$ or $q_2$. We define $f$ so that whenever $\mathcal{A}$ is in state $s$ and reads $a$, it directs the run to proceed as follows. If the run has just started or $s$ was reached from $q_6$ or $q_2$, then the run continues to $q_1$; if $s$ was reached from $q_5$ or $q_1$, then the run continues to $q_2$. First, note that the above strategy can be described by means of a function with domain $\{a, 0, 1\}^*$. For example, $f(a) = q_1$, $f(a0a) = q_2$, and $f(a1a1a) = q_2$. In addition, the strategy guarantees that all words in the language are accepted. Indeed, reading $a0a0$ either leads to a traversal of an $\alpha$-transition or leads to $s$, where the next $a0a0$ or $a1a1$ leads to a traversal of an $\alpha$-transition, and similarly for $a1a1$.

It is left to prove that $\mathcal{A}$ is not DBP. Recall that there are two ways to make $\mathcal{A}$ deterministic by pruning: either prune the $a$ transition from $s$ to $q_1$ or the $a$-transition from $s$ to $q_2$. We show that both ways result in a DBW whose language is strictly contained in that of $\mathcal{A}$. First, if we prune the transition from $s$ to $q_1$, then the obtained tDBW rejects the word $(a1)^\omega$, which is in $L(\mathcal{A})$. Indeed, the singe run on it is $(s, q_2)^\omega$, which is rejecting. Dually, if we prune the transition from $s$ to $q_2$, then the single run of the obtained DBW on the word $(a0)^\omega$, which is in $L(\mathcal{A})$, is $(s, q_1)^\omega$, which is rejecting. Thus, $\mathcal{A}$ is an HD-NBW that is not DBP. □

Note that Theorem 6 does not imply that HD-NBWs are more succinct than DBWs. In fact, the HD-NBW $\mathcal{A}$ in the proof can be easily determinized by merging the states $q_1$ and $q_2$. The succinctness of HD-NBWs with respect to DBWs is an intriguing open problem. It is shown in [13,3] that HD-NBWs can be determinized with a quadratic blow-up, but no matching lower bounds is known. In fact, we have no example to an HD-NBW that is smaller in even one state from an equivalent DBW.

For HD-NCWs, the succinctness picture is clear, and includes a surprising exponential succinctness of HD-NCW with respect to DCWs. The beautiful proof, by Kuperberg and Skrzypczak, shows how the past can direct a nondeterministic automaton how to resolve its nondeterministic choices, and to do it in a way that significantly reduces the state space of the automaton. Below we present the key ideas of the proof.

**Theorem 7.** [13] *There is an infinite family of languages $L_1, L_2, L_3, \ldots$ such that for every $n \geq 1$, the following holds.*

1. *There is an HD-NCW with $2n$ states that recognizes $L_n$.*
2. *Every DCW that recognizes $L_n$ needs at least $\frac{2^n}{2n}$ states.*

*Proof.* For $n \geq 1$, let $[n] = \{0, 1, \ldots, 2n-1\}$. We define the language $L_n$ over the alphabet $\Sigma = \{I, Z, X, H\}$. Each letter in $\Sigma$ is a (possibly partial) function $\sigma : [n] \to [n]$, as described in Figure 4.

The functions $I, X$, and $Z$ are one-one and onto: for every $x \in [n]$, we have that $I(x) = x$, $Z(x) = (x + 1) \bmod 2n - 1$, and $X$ agrees with $I$, except for $x \in \{0, 1\}$, where $X(0) = 1$ and $X(1) = 0$. The function $H$ is partial; it agrees with $I$, except for $x = 0$, where $H(0)$ is undefined. Thus, the letters induce permutations on $[n]$, with $H$ inducing a permutation only on $[n] \setminus \{0\}$.

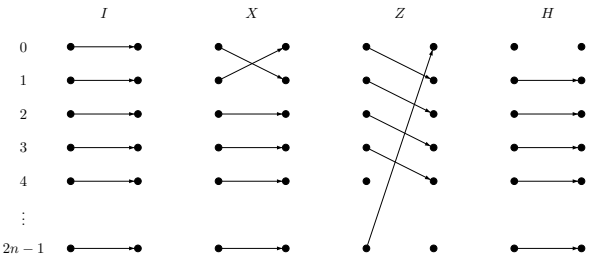


Fig. 4: The permutations induced by the letters $I$, $X$, $Z$, and $H$.

We view a finite word $w$ as the partial function $w : [n] \to [n]$ obtained by composing its letters. Thus, if $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_l$, then for all $i \in [n]$, we have that $w(i) = \sigma_l(\cdots \sigma_2(\sigma_1(i)))$. It is convenient to associate with each word $w \in \Sigma^*$ a grid of dimensions $(|w| + 1) \times 2n$, and lines that start in "floors" in $[n]$ and traverse the floors along the grid according to the permutations induced by the letters in $w$. As $H(0)$ is undefined, a line that reaches floor 0 before $H$ is read has a "hole" in the corresponding position in the grid. Figure 5 describes the grid associated with the word $IXHZZXHZ$ when $n = 2$ and $n = 3$.

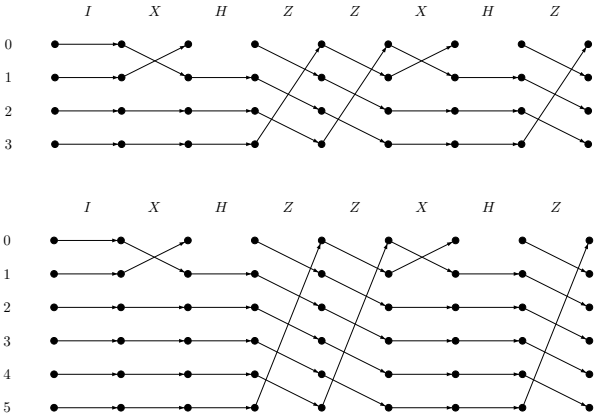


Fig. 5: The grid induced by the word $IXHZZXHZ$ when $n = 2$ (top) and $n = 3$ (bottom).

An infinite word $u \in \Sigma^\omega$ corresponds to an infinite sequence of compositions of its letters, and thus the horizontal dimension of the grid associated with it is infinite. We define $L_n$ as the set of words in $\Sigma^\omega$ whose grid contains an infinite line; that is, a line that has only finitely many holes. For example, back to Figure 5, it is not hard to see that the infinite word $u = w^\omega$, for $w = IXHZZXHZ$

is in $L_2$. Indeed, when $n = 2$, we have that $w(0) = 0$ and $w(2) = 2$, and so the lines starting at floors 0 and 2 are never cut. On the other hand, $u \notin L_3$. Indeed, when $n = 3$, we have that $w(0) = 4$, $w(2) = 5$, $w(3) = 0$, $w(4) = 2$, whereas $w(1)$ and $w(5)$ are undefined. Accordingly, $w^5(i)$ is undefined for all $i \in [3]$, implying that lines from all floors are cut whenever $w^5$ is read. Therefore, the grid of $u$ contains no infinite line, and so $u \notin L_3$.

Intuitively, a DCW for $L_n$ needs exponentially many states as it has to remember in its state space the subset of floors whose lines have not been cut since the last "break point", namely the last time since all floors have been cut. The formal proof is based on combinatorial arguments that are less related to the issue of HDness, and can be found in [13]. Here, we prove the first claim in the theorem, namely the existence of an HD-NCW with $O(n)$ states that recognizes $L_n$.

It is easy to see that $L_n$ can be recognized by an NCW with $2n$ states. Indeed, an NCW can simply guesses a line to follow, and to initiate its guess whenever the line it follows is cut. Specifically (see $\mathcal{A}_2$ in Figure 6), the NCW $\mathcal{A}_n$ has state space $\{q_0, \ldots, q_{2n-1}\}$ and for all $i \in [n]$, it visits $q_i$ when the line it follows is in floor $i$. The initial state of $\mathcal{A}_n$ is arbitrarily set to $q_0$, and the transition function updates the floor according to the letter it reads. For example, when $\mathcal{A}_n$ is in state $q_i$ and reads $I$, it stays in $q_i$, when it reads $X$, if stays in $q_i$ for $i \in \{2, \ldots, 2n-1\}$, moves to $q_1$ from $q_0$, and moves to $q_0$ from $q_1$. Nondeterminism is required when $\mathcal{A}_n$ reads the letter $H$ in state $q_0$, thus when it follows a line that is in floor 0 and the line is cut. Then, $\mathcal{A}_n$ guesses a new floor to follow. Since all floors are candidates for hosting an infinite line, $\mathcal{A}_n$ can nondeterministically move from $q_0$ with $H$ to all states. Since the input word is in the language if it contains an infinite line, thus if it is possible to eventually follow a line that is never cut, we want an accepting run to take only finitely many $H$-transitions from the state $q_0$, thus $\alpha$ is the set of these transitions.

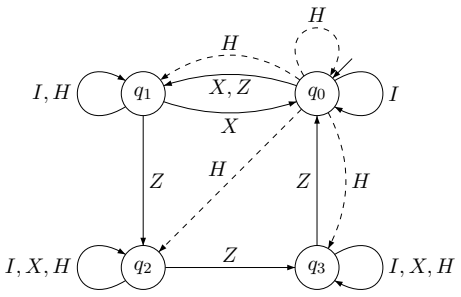


Fig. 6: The HD-NCW $\mathcal{A}_2$ that recognizes $L_2$.

It is less easy to see that the NCW $\mathcal{A}_n$ is in fact HD. In order to see this, consider the following HD strategy $g : \Sigma^* \to Q$. First, $g(\epsilon) = q_0$, thus all runs start in state $q_0$, which is the only initial state of $\mathcal{A}_n$. Whenever a run is in

state $q_0$ after reading a prefix $u$ and it reads $H$, it proceeds to the state $q_i$ such that the line that is now in floor $i$ is the longest among all lines in the graph. Formally, for all words $u \in \Sigma^*$ and floors $i \in [n]$, let $\mathit{seniority}(u, i)$ be the length of the longest suffix $u'$ of $u$ such that there is $j \in [n]$ with $u'(j) = i$. Then, if $u \in \Sigma^*$ is such that $g(u) = q_0$, then $g(u \cdot H) = q_i$, for the minimal $i \in [n]$ that maximizes $\mathit{seniority}(u, i)$. Note that the choice of the minimal $i$ is arbitrary, and it is required in order to decide between lines with the same seniority. Note also that it is possible to implement the HD strategy $g$ by maintaining the order of seniority among the different floors during the run, thus it indeed depends only on the history of the run. Finally, as an infinite line would eventually obtain the maximal seniority, it is guaranteed that following the strategy $g$ leads to accepting all words in the language: in all of then, the run that follows $g$ eventually follows an infinite line. □

The two results we described here about the succinctness of HD-automata with respect to deterministic ones are just two examples to the interesting and extensive research that has been done in the last years about HD automata. Research includes richer types of HD automata, both in terms of the acceptance condition, and in settings like alternating, weighted, or pushdown automata [6,7], the problem of deciding HDness [4], the relation of HDness to other restrictions on nondeterminism [2], minimization and canonization of HD automata [1,20], and many more [14].

## References


1. B. Abu Radi and O. Kupferman. Minimization and canonization of GFG transition-based automata. *Log. Methods Comput. Sci.*, 18(3), 2022.
2. B. Abu Radi, O. Kupferman, and O. Leshkowitz. A hierarchy of nondeterminism. In *46th Int. Symp. on Mathematical Foundations of Computer Science*, volume 202 of *LIPIcs*, pages 85:1–85:21, 2021.
3. B. Abu Radi, O. Kupferman, and O. Leshkowitz. Easy complementation of history-deterministic büchi automata. In *22nd Int. Symp. on Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science. Springer, 2024.
4. M. Bagnol and D. Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *Proc. 38th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 122 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
5. U. Boker, D. Kuperberg, O. Kupferman, and M. Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Proc. 40th Int. Colloq. on Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 89–100, 2013.
6. U. Boker, D. Kuperberg, K. Lehtinen, and M. Skrzypczak. On the succinctness of alternating parity good-for-games automata. In *Proc. 40th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 182 of *LIPIcs*, pages 41:1–41:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
7. U. Boker and K. Lehtinen. When a little nondeterminism goes a long way: an introduction to history-determinism. *ACM SIGLOG News*, 10(1):177–196, 2023.
8. A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.

9. T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proc. 36th Int. Colloq. on Automata, Languages, and Programming*, volume 5556 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2009.
10. R. Faran and O. Kupferman. On $(I/O)$-aware good-for-games automata. In *18th Int. Symp. on Automated Technology for Verification and Analysis*, volume 12302 of *Lecture Notes in Computer Science*, pages 161–178. Springer, 2020.
11. E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
12. T.A. Henzinger and N. Piterman. Solving games without determinization. In *Proc. 15th Annual Conf. of the European Association for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2006.
13. D. Kuperberg and M. Skrzypczak. On determinisation of good-for-games automata. In *Proc. 42nd Int. Colloq. on Automata, Languages, and Programming*, pages 299–310, 2015.
14. O. Kupferman. Using the past for resolving the future. *Frontiers in Computer Science*, 4, 2022.
15. O. Kupferman, D. Sadigh, and S.A. Seshia. Synthesis with clairvoyance. In *7th International Haifa Verification Conference*, volume 7261 of *Lecture Notes in Computer Science*, pages 5–19. Springer, 2011.
16. O. Kupferman, S. Safra, and M.Y. Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006.
17. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(2):408–429, 2001.
18. A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
19. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.
20. S. Schewe. Minimising good-for-games automata is NP-complete. In *Proc. 40th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 182 of *LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
21. A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
22. W. Thomas. On the synthesis of strategies in infinite games. In *Proc. 12th Symp. on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995.
23. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and Systems Science*, 32(2):182–221, 1986.
24. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.