Temporal Synthesis for Bounded Systems and Environments*

Orna Kupferman[1], Yoad Lustig[2], Moshe Y. Vardi[2], and Mihalis Yannakakis[3]

1 **Hebrew University**
2 **Rice University**
3 **Columbia University**

Abstract

Temporal synthesis is the automated construction of a system from its temporal specification. It is by now realized that requiring the synthesized system to satisfy the specifications against all possible environments may be too demanding, and, dually, allowing all systems may be not demanding enough; systems and environments that are too large may not be feasible in practice. In this work we study *bounded temporal synthesis*, in which bounds on the sizes of the state space of the system and the environment are additional parameters to the synthesis problem. This study is motivated by the fact that such bounds may indeed change the answer to the synthesis problem, as well as the theoretical and computational aspects of the synthesis problem. In particular, a finer analysis of synthesis, which takes system and environment sizes into account, yields deeper insight into the quantificational structure of the synthesis problem and the relationship between strong synthesis – there exists a system such that for all environments, the specification holds, and weak synthesis – for all environments there exists a system such that the specification holds.

We first show that unlike the unbounded setting, where determinacy of regular games implies that strong and weak synthesis coincide, these notions do not coincide in the bounded setting. We then turn to study the complexity of deciding strong and weak synthesis. We show that bounding the size of the system or both the system and the environment, turns the synthesis problem into a search problem, and one cannot expect to do better than brute-force search. In particular, the synthesis problem for bounded systems and environment is Σ_2^P-complete (in terms of the bounds, for a specification given by a deterministic automaton). We also show that while bounding the environment may lead to the synthesis of specifications that are otherwise unrealizable, such relaxation of the problem comes at a high price from a complexity-theoretic point of view.

1 Introduction

Temporal synthesis is the automated construction of a system from its temporal specification. The basic idea is simple and appealing: instead of developing a system and verifying that it satisfies its specification, we would like to have an automated procedure that, given a specification, constructs a system that is correct by construction. The first formulation of synthesis goes back to Church [5]; the modern approach was initiated by Pnueli and Rosner, who introduced LTL (linear temporal logic) synthesis [24]. The *LTL synthesis problem* receives as input a specification given by means of an LTL formula and outputs a reactive system modeled by a finite-state transducer satisfying the given specification — if such exists.

In the specification to the system, it is important to distinguish between output signals, controlled by the system, and input signals, controlled by the environment. A system should satisfy its specification against all possible environments. Therefore, the quantification structure on input and output

* Supported in part by NSF grants CCF-0613889, CCF-0728882, CCF-0728736, CCF-1017955, and CNS 1049862, by BSF grant 9800096, and by gift from Intel. Part of this work was done while Moshe Y. Vardi was on sabbatical at the Hebrew University in Jerusalem, supported by a Forchheimer Visiting Professorship.

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

signals is different. Output signals are existentially quantified while input signals are universally quantified [24]. It is by now realized that requiring the synthesized system to satisfy the specification against all possible environments is often too demanding. Dually, allowing all possible systems is perhaps not demanding enough. This issue is traditionally approached by adding assumptions on the system and/or the environment, which are modeled as part of the specification (c.f., [4]). [1]

In this work we study *bounded temporal synthesis*, in which assumptions on the system and its environment are given by means of bounds on the sizes of their state space. Thus, in addition to a specification ψ, the input to the bounded synthesis problem contains two parameters $n, m \geq 1$, and a specification ψ is (n, m)-*realizable*, if there is a transducer $\mathcal{T}$ with n states such that for all transducers $\mathcal{T}'$ with m states, the computation $\mathcal{T} \| T'$ – generated by the interaction of $\mathcal{T}$ with $\mathcal{T}'$, satisfies ψ. Note that traditional synthesis corresponds to the case $n = m = \infty$. [2] Also note that by setting only one of n or m to ∞, we can consider a setting in which only one of the components is bounded. In particular, [27] studies the setting in which only the system is bounded. We note that the need to bound the environment is of interest in several other paradigms in computer science. For example, in cryptography, one studies the security of a given cryptosystem with respect to attackers with bounded (typically polynomial) computational power [20], and in the analysis of on-line algorithms one sometimes care for the competitive ratio of a given on-line algorithm with respect to requests issued by a bounded adversary [2]. Even closer to the work here is the study of bounded rationality in games, where bounds are placed on the power of the players. In particular, having the players be automata with a bounded number of states is a natural way of doing this. As shown in [22], such bounds affect the kind of equilibria one gets, and gives in fact a way of getting around some of the problematic cases of equilibria, (e.g., in the Prisoner's Dilemma [26]).

It is not hard to see that bounding the size of the system or its environment may indeed change the answer to the synthesis problem. Clearly, already in a setting with no interaction, bounding the size of a system may prevent it from satisfying some specifications. In the presence of interaction, bounding the size of the environment both restricts the possible behaviors of the environment and enables the system to "learn" the environment. For example, knowing that the environment has a single state implies that the input to the system is fixed, thus a specification like "if p holds in the present, then p holds always", for an input signal p, is realizable against environments with a single state, while it is clearly not realizable in general.

Traditional temporal synthesis is *determined*, in the sense that for every specification ψ, either there is a system that realizes ψ, or there is an environment that realizes $\neg\psi$. Note that not having a system that realizes ψ only means that for every system $\mathcal{T}$, there is an environment $\mathcal{T}'$ such that the computation $\mathcal{T} \| \mathcal{T}'$ does not satisfy ψ. This by itself does not imply that there is an environment $\mathcal{T}'$ such that for all systems $\mathcal{T}$, the computation $\mathcal{T} \| \mathcal{T}'$ satisfies $\neg\psi$. However, by determinacy of Borel games [19], we know that the lack of $\mathcal{T}$ that realizes ψ does imply the existence of $\mathcal{T}'$ that realizes $\neg\psi$. We show that determinacy no longer holds in the bounded setting. In particular, for every $k \geq 1$ there is a specification ψ_k such that ψ_k cannot be realized against environments of size k, nor can $\neg\psi_k$ be realized by an environment of size k.

The observation about determinacy, which uses the theory of *checking sequences* for transducers [18], yields deeper insight into the quantificational structure of the synthesis problem and the relationship between two possible definitions of synthesis in the bounded setting: *strong synthesis*, where there is a system $\mathcal{T}$ such that for all environments $\mathcal{T}'$, the computation $\mathcal{T} \| \mathcal{T}'$ satisfies the

[1] A different, more conceptual, way to restrict the range of environments with respect to which the system has to satisfy the specification is to assume that the environment has objectives of its own, and is therefore *rational*, rather than hostile [12].

[2] In fact, by the small model property, already to the case n and m are doubly exponential in the length of $|\psi|$ [8, 24].

specification, and *weak synthesis*, where for all environments $\mathcal{T}'$ there exists a system $\mathcal{T}$ such that the computation $\mathcal{T}\|\mathcal{T}'$ satisfies the specification. In contrast, in the unbounded setting, strong and weak synthesis coincide.

We study the complexity of strong and weak synthesis. Recall that LTL synthesis is 2EXPTIME-complete [24]. This high complexity, along with technical challenges in the implementation of synthesis algorithms [14, 33], have led researchers to develop synthesis algorithms for fragments of LTL or alternative specification formalisms [17, 23]. A different approach for coping with the complexity of LTL synthesis, tightly related to our work here, is to restrict attention to systems of a bounded size [27]. As argued in [27], bounding the size of the system enables a reduction of the synthesis problem to the SAT problem, and also leads to the decidability of synthesis of distributed systems. For the bounded setting, researchers were also able to come up with a symbolic implementation [7, 10]. [3]

Recall that the bounded synthesis problem has three parameters: ψ, n, and m. We would like to study the complexity in terms of each of n and m. One standard way to analyze the complexity in terms of one parameter is to fix the other parameters. This standard way does not work in our setting, as fixing ψ implies fixing also n and m. Indeed, by [8, 24], if ψ is realizable, then it is also realizable by a transducer with doubly-exponentially many states. Also, by determinacy, ψ is not realizable if the environment can realize $\neg\psi$ by a transducer that is doubly-exponential in the length of ψ. Accordingly, we have to neutralize the dominance of ψ in the complexity analysis in a different way. We do so by assuming that the temporal specification is given, instead of as an LTL formula, as a deterministic Büchi automaton. For such specifications, the unbounded realizability problem amounts to checking the nonemptiness of a deterministic Büchi automaton, and can therefore be solved in quadratic time [31]. In Section 4, we justify the choice of deterministic Büchi automata further.

We first show that bounding the size of the system enables an easy reduction from the synthesis problem to the model-checking problem. At the same time, one cannot expect to do better than brute-force search in synthesizing bounded-size systems. Formally, we show that deciding whether a specification $\mathcal{A}$ givens by means of a deterministic Büchi automaton is realizable by a system with n states is NP-complete, for n given in unary. The proof of NP-hardness is technically easy and is similar to known NP-hardness proofs in the context of formal methods [6, 15]. Still, it justifies the reduction to SAT given in [27] without a lower bound, and it sets the stage to the much more challenging lower bound, for the case both the system and the environment are bounded: we show that deciding whether a specification $\mathcal{A}$ is realizable by a system with n states against all environment with m states is Σ_2^P-complete, for n and m given in unary. Thus, brute-force search is the best we can do also here, and one cannot expect to do better than model checking the interaction of all systems with n states with all environments with m states.

We also show that while bounding the environment may indeed lead to the synthesis of specifications that are otherwise unrealizable, such relaxation of the problem comes at a high price from a complexity-theoretic point of view. As pointed above, synthesis with respect to specifications given by deterministic Büchi automata is quadratic. Adding a bound on the size of the environment seems to add a cost that is doubly exponential in that size. In fact, we show that even model checking against bounded environments is apparently harder than standard model checking. Finally, we formalize the intuition of the system being able to learn the bounded environment by showing that for absolute liveness properties, which are insensitive to additions of prefixes [28], weak and strong realizability

[3] As mentioned above, [27] also studies synthesis of bounded systems. The contributions here and in [27] are, however, different. In [27], the focus is on practical algorithm for the setting of a bounded system. Here, the focus is on the theoretical aspects of the problem and its complexity, and rather than studying bounded systems, we consider bounds on the system, the environment, and both.

coincide, and the complexity of the problem is only exponential in the bound for the environment.

2 Preliminaries

Consider finite sets I and O of input and output signals, respectively. We model finite-state reactive systems with inputs in I and outputs in O by *transducers* (I/O-transducers, when I and O are not clear from the context). A transducer is a finite graph with a designated start state, where the edges are labeled by letters in 2^I and the states are labeled by letters in 2^O. Formally, a transducer is a tuple $\mathcal{T} = \langle I, O, S, s_{in}, \eta, L\rangle$, where I and O are the sets of input and output signals, S is a finite set of states, $s_{in} \in S$ is an initial state, $\eta : S \times 2^I \to S$ is a deterministic transition function, and $L : S \to 2^O$ is a labeling function. We extend η to words in $(2^I)^*$ in the straightforward way. Thus, $\eta : (2^I)^* \to S$ is such that $\eta(\varepsilon) = s_{in}$, and for $x \in (2^I)^*$ and $i \in 2^I$, we have $\eta(x \cdot i) = \eta(\eta(x), i)$. Each transducer $\mathcal{T}$ induces a *strategy* $f_{\mathcal{T}} : (2^I)^* \to 2^O$ where for all $w \in (2^I)^*$, we have $f_{\mathcal{T}}(w) = \tau(L(w))$. Thus, $f_{\mathcal{T}}(w)$ is the letter that $\mathcal{T}$ outputs after reading the sequence w of input letters. A transducer with at most k states is referred to as a *k-transducer*.

Consider an infinite sequence $w = i_0, i_1, i_2, i_3, \ldots \in (2^I)^\omega$ of input letters. The computation of $\mathcal{T}$ on w, denoted $\mathcal{T}(w)$, is $\rho = (o_0 \cup i_0), (o_1 \cup i_1), (o_2 \cup i_2), \ldots \in (2^{I \cup O})^\omega$ such that for all $j \geq 0$, we have $o_j = f_{\mathcal{T}}(i_0 \cdot i_1 \cdots i_{j-1})$. Note that, in particular, $o_0 = f_{\mathcal{T}}(\epsilon)$. Thus, the mode of interaction we assume is that the transducer initiates the interaction with the environment by outputting $f_{\mathcal{T}}(\epsilon)$, the environment then responds with i_0 (making $f_{\mathcal{T}}(\epsilon) \cup i_0$ the set of signals that are valid in the first time unit), then the transducer responds with $f_{\mathcal{T}}(i_0)$, the environment with i_1, and so on. In order to emphasize the fact that the transducer moves first, we sometimes refer to ρ as a *1-computation* of $\mathcal{T}$. Also, we sometimes refer to ρ as w/y, for $y = o_0, o_1, o_2 \ldots$.

One could also consider a dual type of interaction, in which the environment moves first. Then, a *2-computation* of $\mathcal{T}$ is $\rho = (o_0 \cup i_0), (o_1 \cup i_1), (o_2 \cup i_2), \ldots$ where for all $j \geq 0$, we have $o_j = f_{\mathcal{T}}(i_0 \cdot i_1 \cdots i_j)$. In particular, $o_0 = f_{\mathcal{T}}(i_0)$. Note that when two transducers interact with each other, one of them initiates the interaction and moves first, thus its computations are 1-computations, and the second moves second, and its computations are 2-computations. We say that $\rho \in (2^{I \cup O})^\omega$ is a computation of $\mathcal{T}$ if there is $w \in (2^I)^\omega$ such that $\rho = \mathcal{T}(w)$. Finally, we sometimes refer also to finite sequences of input letters and the finite computations of $\mathcal{T}$ on them.

We specify on-going behaviors of I/O-transducers by means of LTL formulas over the set $I \cup O$ of atomic propositions, or automata on infinite words over the alphabet $2^{I \cup O}$. We refer to elements in both formalisms as specifications over $I \cup O$. For a specification ψ over $I \cup O$ and a "who moves first" flag $b \in \{1, 2\}$, we use $real_{I,O,b}(\psi)$ to indicate that there is an I/O transducer $\mathcal{T}$ such that all the b-computations of $\mathcal{T}$ satisfy ψ. Given a specification ψ over the sets I and O of input and output signals, the *realizability problem* for ψ is to decide whether $real_{I,O,1}(\psi)$ [24]. For $b \in \{1, 2\}$ let $\tilde{b}$ dualize b (that is, $\tilde{b} = 3 - b$). By determinacy of Borel games [19], we have the following (see also [9]).

▶ **Theorem 1.** *For every specification ψ, precisely one of $real_{I,O,b}(\psi)$ or $real_{O,I,\tilde{b}}(\neg\psi)$ holds.*

3 Strong and Weak realizability

The traditional definition of realizability requires ψ to be satisfied in all the computations of $\mathcal{T}$, ignoring the ability of the environment to feasibly generate the sequences of input letters that induce these computations. In this work, we are interested in realizability with components of a bounded size. In order to define this setting, let us first formalize the interaction between two transducers. For an I/O-transducer $\mathcal{T}$ that induces a strategy $f_{\mathcal{T}} : (2^I)^* \to 2^O$ and an O/I-transducer $\mathcal{T}'$ that induces a strategy $f_{\mathcal{T}'} : (2^O)^* \to 2^I$, the single 1-computation of $\mathcal{T} \| \mathcal{T}'$ is $\rho = (o_0 \cup i_0), (o_1 \cup i_1), (o_2 \cup i_2), \ldots$

where for all $j \geq 0$, we have $o_j = f_{\mathcal{T}}(i_0 \cdot i_1 \cdots i_{j-1})$ and $i_j = f_{\mathcal{T}'}(o_0 \cdot o_1 \cdots o_j)$. The single 2-computation of $\mathcal{T}\|\mathcal{T}'$ is $\rho = (o_0 \cup i_0), (o_1 \cup i_1), (o_2 \cup i_2), \ldots$ where for all $j \geq 0$, we have $i_j = f_{\mathcal{T}'}(o_0 \cdot o_1 \cdots o_{j-1})$ and $o_j = f_{\mathcal{T}}(i_0 \cdot i_1 \cdots i_j)$.

Recall that a specification ψ is realizable if there is an I/O-transducer $\mathcal{T}$ such that all the 1-computations of $\mathcal{T}$ satisfy ψ. It is known that if a transducer $\mathcal{T}$ does have a computation that violates ψ, then $\mathcal{T}$ also has a computation that violates ψ and is induced by an input sequence that is generated by a transducer [3]. Accordingly, ψ is realizable iff there is an I/O-transducer $\mathcal{T}$ such that for all O/I transducers $\mathcal{T}'$, the 1-computation of $\mathcal{T}\|\mathcal{T}'$ satisfies ψ. In Definition 2 below, we call this "strong realizability" and introduce also a weaker type of realizability.

▶ **Definition 2.** [*Strong and Weak Realizability*] Consider a specification ψ over $I \cup O$, a first-move flag $b \in \{1, 2\}$, and $m, n \in \mathcal{N} \cup \{\infty\}$.

- We say that ψ is *strongly (I, O, b)-realizable with respect to systems with n states and environments with m states*, denoted $s\_real_{I,O,b}(\psi, n, m)$, if there is an I/O-transducer $\mathcal{T}$ with at most n states such that for every O/I-transducer $\mathcal{T}'$ with at most m states, the b-computation of $\mathcal{T}\|\mathcal{T}'$ satisfies ψ.
- We say that ψ is *weakly (I, O, b)-realizable with respect to systems with n states and environments with m states*, denoted $w\_real_{I,O,b}(\psi, n, m)$, if for every O/I-transducer $\mathcal{T}'$ with at most m states, there is an I/O-transducer $\mathcal{T}$ with at most n states such that the b-computation of $\mathcal{T}\|\mathcal{T}'$ satisfies ψ.

Strong realizability means that the system can defeat all environment's strategies under consideration. Weak realizability means that the environment does not have a strategy to defeat all the strategies of the system. When strong realizability is impossible, a designer may settle for a weak one. Also, weak realizability of $\neg\psi$ by the environment explains why ψ is not realizable by the system [11]. Formally, we have the following.

▶ **Lemma 3.** $w\_real_{I,O,b}(\psi, n, m)$ *iff not* $s\_real_{O,I,\tilde{b}}(\neg\psi, m, n)$.

As discussed above, $real_{I,O,b}(\psi)$ iff $s\_real_{I,O,b}(\psi, \infty, \infty)$. In fact, as we state below, weak and strong realizability coincide in the unbounded setting.

▶ **Theorem 4.** *For every specification ψ, we have that $w\_real_{I,O,b}(\psi, \infty, \infty)$ iff $s\_real_{I,O,b}(\psi, \infty, \infty)$.*

Proof. By definition, $s\_real_{I,O,b}(\psi, \infty, \infty)$ implies $w\_real_{I,O,b}(\psi, \infty, \infty)$. For the other direction, assume that $s\_real_{I,O,b}(\psi, \infty, \infty)$ does not hold. Then, by Theorem 1, we have that $s\_real_{O,I,\tilde{b}}(\neg\psi, \infty, \infty)$. Thus, by Lemma 3, we have that $w\_real_{I,O,b}(\psi, \infty, \infty)$ does not hold, and we are done. ◀

▶ **Example 5.** Let $I = \{p\}$, $O = \{q\}$, and $\psi = G(q \leftrightarrow Xp)$. Note that the specification requires the next value of the input signal p to depend on the current value of the output signal q. Since the system has no control on the input signals, we have that not $s\_real_{I,O,1}(\psi, \infty, \infty)$, and in fact even not $s\_real_{I,O,1}(\psi, \infty, 1)$. To formally prove this, we use Lemma 3 and show that $w\_real_{O,I,2}(\neg\psi, 1, \infty)$. Note that $\neg\psi = F((q \wedge X\neg p) \vee (\neg q \wedge Xp))$. Now, for every I/O-transducer $\mathcal{T}$, if $\mathcal{T}$ outputs q in its initial state, the O/I-1-transducer $\mathcal{T}'$ that always outputs $\neg p$ is such that the unique 2-computation of $\mathcal{T}'\|\mathcal{T}$ satisfies $\neg\psi$. Similarly, if $\mathcal{T}$ outputs $\neg q$ in its initial state, the corresponding O/I-1-transducer $\mathcal{T}'$ always outputs p.

On the other hand, $w\_real_{I,O,1}(\psi, \infty, 1)$. Indeed, there are two possible O/I-1-transducers: $\mathcal{T}'_1$ that always outputs p, and $\mathcal{T}'_2$ that always outputs $\neg p$. For $\mathcal{T}'_1$, the I/O-transducer $\mathcal{T}$ that always outputs q is such that the unique 1-computation of $\mathcal{T}\|\mathcal{T}'_2$ is $\{p, q\}^\omega$. For $\mathcal{T}'_2$, the I/O-transducer $\mathcal{T}$ that always outputs $\neg q$ is such that the unique 1-computation of $\mathcal{T}\|\mathcal{T}'_2$ is $\emptyset^\omega$. Since both satisfy ψ, we are done.

Example 5 shows that strong and weak realizability do not coincide in the bounded setting. We now generalize the example to all bounds, and show that it implies that the games that correspond to bounded synthesis are not determined. We first need some notations.

For two I/O-transducers $\mathcal{T}_1$ and $\mathcal{T}_2$, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *equivalent*, denoted $\mathcal{T}_1 \equiv \mathcal{T}_2$, if $\mathcal{T}_1(w) = \mathcal{T}_2(w)$ for every word $w \in (2^I)^*$; otherwise, $\mathcal{T}_1$ and $\mathcal{T}_2$ are inequivalent. If $\mathcal{T}$ is a transducer and s an state of $\mathcal{T}$, then denote by $\mathcal{T}/s$ the transducer that is the same as $\mathcal{T}$ except that it has s as the start state. We say that $\mathcal{T}$ is *minimized* if there is no transducer $\mathcal{T}'$ such that $\mathcal{T} \equiv \mathcal{T}'$ and $\mathcal{T}'$ has strictly fewer states than $\mathcal{T}$. We say that two transducers $\mathcal{T}_1, \mathcal{T}_2$ are *structurally equivalent*, denoted $\mathcal{T}_1 \sim \mathcal{T}_2$, if (i) for every state s_1 of $\mathcal{T}_1$ there is a state s_2 of $\mathcal{T}_2$ such that $\mathcal{T}_1/s_1 \equiv \mathcal{T}_2/s_2$, and (ii) conversely, for every state s_2 of $\mathcal{T}_2$ there is a state s_1 of $\mathcal{T}_1$ such that $\mathcal{T}_1/s_1 \equiv \mathcal{T}_2/s_2$. Note that two minimized transducers are structurally equivalent iff they are isomorphic (but their start states do not need to map to each other in the isomorphism) [18].

Consider an I/O-transducer $\mathcal{T}$ with k states. A *checking sequence* for $\mathcal{T}$ is a word $w \in (2^I)^*$ such that for all transducers $\mathcal{T}'$ with at most k states, if $\mathcal{T}'(w) = \mathcal{T}(w)$ then $\mathcal{T}' \sim \mathcal{T}$. A transducer is *strongly connected* if every state can reach every other state. It is known that every strongly connected k-state transducer $\mathcal{T}$ has a checking sequence (of length at most exponential in k), and furthermore, if $\mathcal{T}$ is minimized, then it has one of length polynomial in k; see [18] for background and an overview on checking and other test sequences of transducers.

For a word $w \in (2^I)^*$, let $\theta(w)$ be an LTL formula over I that holds in exactly all computations in $(2^{I \cup O})^\omega$ whose prefix agrees with w on the signals in I. Thus, if $w = i_0, i_1, \ldots, i_l$, then $\theta(w) = \bigwedge_{0 \leq j \leq l} X^j((\wedge_{x \in i_j} x) \wedge (\wedge_{x \notin i_j} \neg x))$. We define $\theta(y)$ similarly, for $y \in (2^O)^*$.

We now use checking sequences in order to show that bounding the sizes of the components, strong and weak synthesis no longer coincide, and determinacy fails.

▶ **Theorem 6.** *For every $k \geq 1$ there is an LTL formula ψ_k such that $w\_real_{I,O,1}(\psi_k, \infty, k)$ but not $s\_real_{I,O,1}(\psi_k, \infty, k)$; equivalently, neither $s\_real_{I,O,1}(\psi_k, \infty, k)$ nor $s\_real_{I,O,2}(\neg\psi_k, k, \infty)$.*

Proof. We give first a high level idea of the approach. We construct an LTL formula ψ_k that is weakly realizable, and is "almost strongly realizable", in the sense that the system can succeed in ensuring the formula by using one of two strategies (transducers): the first strategy works for all environment k-transducers except for those that are isomorphic to a specific one $\mathcal{T}_1'$, and the second strategy works for all environment k-transducers except for those isomorphic to a second one $\mathcal{T}_2'$. However, no system strategy works for both $\mathcal{T}_1', \mathcal{T}_2'$, and thus if the system does not know the environment transducer, then it cannot ensure ψ_k, i.e. ψ_k is not strongly realizable. In our construction, we pick suitable transducers $\mathcal{T}_1'$ and $\mathcal{T}_2'$ and construct from them a formula ψ_k that has the desired properties. Note that the presence of a bound on the size of the transducers is critical for this to be possible (and is obviously essential for the construction), i.e. there is no such formula ψ_∞, since weak and strong realizability coincide in the unbounded case.

We proceed now with the details of the construction. For an O/I transducer $\mathcal{T}_i'$ with k states that is minimized and strongly connected, let $y_i \in (2^O)^*$ be a checking sequence for $\mathcal{T}_i'$, let y_i/w_i be the 2-computation of $\mathcal{T}_i'$ on y_i, and let $\varphi_i = \theta(y_i) \wedge \neg\theta(w_i)$. Note that $s\_real_{I,O,1}(\varphi_i, \infty, k-1)$. To see why, consider an I/O-transducer $\mathcal{T}$ that ignores the input and outputs y_i. Since y_i is a checking sequence for $\mathcal{T}_i'$, and $\mathcal{T}_i'$ has k states and is minimized, the sequence y_i/w_i cannot be generated in an interaction with an O/I-transducer with at most k states that is not isomorphic with $\mathcal{T}_i'$. Hence, for every transducer $\mathcal{T}'$ with $k-1$ states, the 1-computation of $\mathcal{T} \| \mathcal{T}'$ satisfies both $\theta(y_i)$ and $\neg\theta(w_i)$. On the other hand, note also that $\neg s\_real_{I,O,1}(\varphi_i, \infty, k)$ and even $\neg w\_real_{I,O,1}(\varphi_i, \infty, k)$. Indeed, the O/I-transducer $\mathcal{T}_i'$ is such that for all I/O-transducers $\mathcal{T}$, if the 1-computation of $\mathcal{T} \| \mathcal{T}_i'$ satisfies $\theta(y_i)$, then it also satisfies $\theta(w_i)$.

We would like to use φ_i in order to construct a specification ψ_k as required. Let $\mathcal{T}_1'$ and $\mathcal{T}_2'$ be two nonisomorphic minimized strongly connected O/I transducers with k states; thus $\mathcal{T}_1'$, $\mathcal{T}_2'$

are not structurally equivalent (since they are minimized). Let o_1 and o_2 be two different letters in 2^O, let s_1 be the state that $\mathcal{T}'_1$ reaches when it reads o_1 from its initial state, and let s_2 be the state that $\mathcal{T}'_2$ reaches when it reads o_2 from its initial state. Finally, let y_1 be a checking sequence for the transducer $\mathcal{T}'_1/s_1$ (i.e., $\mathcal{T}'_1$ with initial state s_1) with corresponding output w_1, and let y_2 be a checking sequence for $\mathcal{T}'_2/s_2$ with corresponding output w_2. We define $\psi_k = (o_1 \wedge X(\theta(y_1) \wedge \neg\theta(w_1))) \vee (o_2 \wedge X(\theta(y_2) \wedge \neg\theta(w_2)))$.

We prove that $w\_real_{I,O,1}(\psi_k, \infty, k)$ but $\neg s\_real_{I,O,1}(\psi_k, \infty, k)$.

We first prove that $w\_real_{I,O,1}(\psi_k, \infty, k)$ holds. Given an O/I-transducer $\mathcal{T}'$ with at most k states, if $\mathcal{T}'$ is not isomorphic to $\mathcal{T}'_1$, then let $\mathcal{T}$ be the I/O-transducer that ignores the input and generates $o_1 \cdot y_1$. Since $\mathcal{T}'_1/s_1$ is a minimized k-state transducer with checking sequence y_1, and $\mathcal{T}'$ is not isomorphic to $\mathcal{T}'_1/s_1$, we know that the output of $\mathcal{T}'$, starting from any state, on input y_1 is not w_1. Therefore, the 1-computation of $\mathcal{T}\|\mathcal{T}'$ satisfies the $o_1 \wedge X(\theta(y_1) \wedge \neg\theta(w_1))$ disjunct of ψ_k. On the other hand, if $\mathcal{T}'$ is isomorphic to $\mathcal{T}'_1$, then $\mathcal{T}'$ is not isomorphic to $\mathcal{T}'_2$. In this case we let $\mathcal{T}$ be the I/O-transducer that ignores the input and generates $o_2 \cdot y_2$. The 1-computation of $\mathcal{T}\|\mathcal{T}'$ now satisfies the $o_2 \wedge X(\theta(y_2) \wedge \neg\theta(w_2))$ disjunct of ψ_k.

It is left to prove that $s\_real_{I,O,1}(\psi_k, \infty, k)$ does not hold. Assume by way of contradiction that $\mathcal{T}$ is an I/O-transducer that realizes ψ_k. Let o be the output in the initial state of $\mathcal{T}$. In order for the 1-computation of $\mathcal{T}\|\mathcal{T}'_1$ and $\mathcal{T}\|\mathcal{T}'_2$ to satisfy ψ_k, it must be that $o = o_1$ or $o = o_2$. If $o = o_1$, then the disjunct of ψ_k that may be satisfied is $o_1 \wedge X(\theta(y_1) \wedge \neg\theta(w_1))$. However, by the definition of y_1 as a checking sequence for $\mathcal{T}'_1/s_1$ with w_1 as the corresponding output, and the fact that $\mathcal{T}'_2$ is not isomorphic to $\mathcal{T}'_1$, it follows that the 1-computation of $\mathcal{T}\|\mathcal{T}'_2$ does not satisfy $X(\theta(y_1) \wedge \neg\theta(w_1))$. If $o = o_2$ we proceed dually with the second disjunct and the 1-computation of $\mathcal{T}\|\mathcal{T}'_1$. ◀

4 Bounded Systems

We start by studying the strong-realizability problem for bounded systems. This problem is first studied in [27]. The motivation there is to deal with the computational complexity of temporal synthesis: the complexity of deciding whether $real_{I,O,b}(\psi)$ holds is known to be 2EXPTIME-complete [24]. Furthermore, for each $n > 0$ one can construct an LTL formula φ_n of length $O(n)$ such that the smallest transducer $\mathcal{T}$ realizing φ_n has at least 2^{2^n} states [25]. In practice, therefore, we may want to bound the size of the systems under consideration, motivating us to ask whether $s\_real_{I,O,b}(\psi, k, \infty)$ holds, instead of asking whether $real_{I,O,b}(\psi)$ holds.

Note that the strong realizability problem for bounded systems has two parameters: a specification ψ and a bound k. Here we would like to understand the complexity with respect to k. Note also that even when one is provided with a candidate transducer $\mathcal{T}$, checking whether $\mathcal{T}$ satisfies an LTL formula ψ is PSPACE-complete with respect to ψ [29]. Thus, we would like to "neutralize" here the effect of ψ on the complexity of checking whether $s\_real_{I,O,b}(\psi, k, \infty)$ holds. Fixing ψ would not help us, as, by the small model property, it induces a fixed bound on the size of a realizing transducer for ψ, if one exists. The complexity of the unbounded synthesis problem for LTL follows from the need to translate the LTL specification to a deterministic automaton on infinite words. Such a translation involves a doubly exponential blow-up [16], and is the source of the complexity of the synthesis problem. Indeed, for specifications given by means of nondeterministic or deterministic Büchi automata, the synthesis problem is complete in EXPTIME and PTIME, respectively [31].

Accordingly, we neutralize the dominance of the specification ψ by considering, instead of a specification given by an LTL formula, a specification given by a deterministic Büchi automaton $\mathcal{A}$ over the alphabet $2^{I \cup O}$. We denote classes of automata by acronyms in $\{D, N\} \times \{F, B\} \times \{W, T\}$. The first letter stands for the branching mode of the automaton (deterministic or nondeterministic); the second letter stands for the acceptance-condition type (finite words or Büchi); the third letter

stands for the object over which the automaton runs (words or trees). For example, NFW stands for nondeterministic automata on finite words, and DBT stands for deterministic Büchi tree automata. We note that while DBWs are less expressive than NBWs, we still work with DBW rather than, say, deterministic parity word automata. The reason is that the nonemptiness problem for deterministic parity tree automata is not known to be polynomial, and we want to emphasize the fact that the hardness results we are going to prove are not due to the automaton and are due to k; the upper bounds we are going to present for DBWs are valid also for specifications given by a deterministic parity word automata.

Working with specifications that are automata, it is convenient to talk about alphabets Σ_I and Σ_O, where the transducers are Σ_I/Σ_O-transducers, in which the transitions are labeled by letters in Σ_I and the states are labeled by letters in Σ_O (or dually, are Σ_O/Σ_I-transducers). The alphabet of the specification DBW is then $\Sigma_O \times \Sigma_I$ when we specify 1-computations, and is $\Sigma_I \times \Sigma_O$ when we specify 2-computations.

▶ **Theorem 7.** *Deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}, k, \infty)$, for $b \in \{1, 2\}$, is NP-complete.*

Proof. We prove the claim for $b = 1$. The argument for $b = 2$ is analogous. We start with membership in NP. Consider a specification $\mathcal{A}$ and a bound $k \geq 1$. A witness that $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A}, k, \infty)$ holds is a Σ_I/Σ_O-transducer $\mathcal{T}$ with k states. All the 1-computations of $\mathcal{T}$ satisfy $\mathcal{A}$ iff the language of $\mathcal{T}$ is contained in the language of $\mathcal{A}$. Since $\mathcal{A}$ is deterministic and can therefore be complemented in linear time, the above containment can be checked in time linear in $\mathcal{T}$ and $\mathcal{A}$ [30].

For the lower bound, we do a reduction from Vertex Cover [13]. Given a graph $G = \langle V, E\rangle$ and $k \geq 1$, we let $\Sigma_I = E$, $\Sigma_O = V$, and we construct a DBW $\mathcal{A}$ of size polynomial in $|V|$ and $|E|$ such that $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A}, k, \infty)$ iff there is $C \subseteq V$ such that $|C| \leq k$ and for every edge $\langle u, v\rangle \in E$, we have $u \in C$ or $v \in C$. Given G, we define $\mathcal{A}$ so that a word $\langle v_0, e_0\rangle, \langle v_1, e_1\rangle, \ldots \in (\Sigma_O \times \Sigma_I)^\omega$ is in $L(\mathcal{A})$ iff for all $j > 0$, we have that v_j is a vertex of e_{j-1}. Formally, $\mathcal{A} = \langle \Sigma_I \times \Sigma_O, E \cup \{q_0\}, q_0, \delta, E\rangle$, where for all $\langle u, v\rangle \in E$ and $\langle t, \langle v', u'\rangle\rangle \in \Sigma_O \times \Sigma_I$, we have that $\delta(q_0, \langle t, \langle u', v'\rangle\rangle) = \langle u', v'\rangle$, and $\delta(\langle u, v\rangle, \langle t, \langle u', v'\rangle\rangle)$ is $\langle u', v'\rangle$ if $t \in \{v, u\}$ and is undefined otherwise. Note that $\mathcal{A}$ is polynomial in $|V|$ and $|E|$. We claim that $\mathcal{A}$ is realizable by a transducer with at most k states iff G has a vertex cover with at most k vertices. Note that we consider here 1-computations, so the transducer receives e_{j-1} as input and then it outputs v_j.

Assume first that G has a vertex cover $C \subseteq V$ with at most k vertices. That is, there is a function $g : E \to C$ such that for every vertex $\langle u, v\rangle \in E$, we have that $g(E) \in \{u, v\}$. A transducer that realizes $\mathcal{A}$ then has C as its state set, where the output of a state v is simply v itself. Upon reading an edge $\langle v, u\rangle$, the transducer moves to the vertex $g(\langle v, u\rangle)$.

Assume now that $\mathcal{A}$ is realized by a transducer $\mathcal{T} = \langle E, V, S, s_0, \rho, \tau\rangle$ such that $|S| \leq k$. Let $C = \{v : v = \tau(s) \text{ for some } s \in S\}$ be the set of vertices of G that are output by states of $\mathcal{T}$. We claim that C is a vertex cover of size at most k for G. Clearly, since $|S| \leq k$, also $|C| \leq k$. Assume by way of contradiction that C is not a vertex cover. Let $\langle u, v\rangle \in E$ be such that $u \notin C$ and $v \notin C$. Since no state in $\mathcal{T}$ outputs u or v, the 1-computation of $\mathcal{T}$ on an input sequence that starts with the letter $\langle u, v\rangle$ does not satisfy $\mathcal{A}$, and we have reached a contradiction. ◀

By Lemma 3, we have that $w\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}, \infty, k)$ iff not $s\_real_{\Sigma_O,\Sigma_I,\tilde{b}}(\mathcal{A}, k, \infty)$. Theorem 7 then immediately implies the following.

▶ **Corollary 8.** *Deciding $w\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}, \infty, k)$, for $b \in \{1, 2\}$, is co-NP-complete.*

The implication of Theorem 7 and Corollary 8 is that deciding strong realizability with bounded systems (resp., weak realizability of bounded environments) amounts to search for a bounded system (resp., environment) that satisfies (resp., falsifies) the specification. Thus, what we have is essentially exhaustive search combined with model checking. The (co)-NP lower bounds tell us that there is no

way of getting around the need to do an exhaustive search. Note that Corollary 8 already refers to the setting in which the environment is bounded, and it studies weak realizability there. In Section 5 below we study strong realizability in this setting.

5 Bounded Environments

We now turn to study the case of strong realizability in a setting of bounded environments (or, dually, weak realizability in a setting of bounded systems). In fact, bounding the environment is of interest already in the context of model-checking. For a system modeled by an I/O-transducer $\mathcal{T}$, a specification $\mathcal{A}$ given by a DBW, and $k \geq 1$, we say that $\mathcal{T}$ satisfies $\mathcal{A}$ with respect to k-environments if for all O/I-k-transducers $\mathcal{T}'$, the computation $\mathcal{T}\|\mathcal{T}'$ satisfies $\mathcal{A}$.

▶ **Theorem 9.** *Given an I/O-transducer $\mathcal{T}$, a DBW $\mathcal{A}$ over the alphabet $2^{I \cup O}$, and $k \geq 1$, the problem of deciding whether $\mathcal{T}$ satisfies $\mathcal{A}$ with respect to k-environments is co-NP-complete.*

Proof. A witness to the fact that $\mathcal{T}$ does not satisfy $\mathcal{A}$ with respect to k-environments is a k-transducer $\mathcal{T}'$ such that $\mathcal{T}\|\mathcal{T}'$ does not satisfy $\mathcal{A}$. The latter can be checked in polynomial time [30].

For the lower bound, we do a reduction from the complement of the Hamiltonian Circle problem [13].[4] Given a graph $G = \langle V, E \rangle$ with k nodes, the idea is to let $\mathcal{T}$ model the graph ($\Sigma_I = \Sigma_O = V$ and, reading v, the transducer $\mathcal{T}$ goes to a vertex whose output is v), and use $\mathcal{A}$ to require that some of the vertices do not appear in the computation. An input sequence from the environment then induces a path in G. It can be shown that G does *not* have a Hamiltonian path iff every k-transducer $\mathcal{T}'$ is such that $\mathcal{T}\|\mathcal{T}'$ satisfies $\mathcal{A}$. ◀

We now turn to study the strong realizability problem when both the system and the environment are bounded.

▶ **Theorem 10.** *Deciding $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A}, n, m)$ and $s\_real_{\Sigma_I,\Sigma_O,2}(\mathcal{A}, n, m)$, for a given DBW $\mathcal{A}$ over alphabets Σ_I, Σ_O, and positive integers n, m in unary, is Σ_2^P-complete.*

Proof. We prove the claim for $b = 1$. The argument for $b = 2$ is analogous. The strong realizability property $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A}, n, m)$ holds iff there exists a Σ_I/Σ_O transducer $\mathcal{T}$ with at most n states such that for every Σ_O/Σ_I transducer $\mathcal{T}'$ with at most m states, the 1-execution of $\mathcal{T}\|\mathcal{T}'$ satisfies the DBW specification $\mathcal{A}$. Membership in Σ_2^P follows from the fact that the sizes of the existentially and universally quantified transducers $\mathcal{T}$ and $\mathcal{T}'$ respectively are polynomially bounded in the size of the input, and the fact that we can check in polynomial time whether the 1-execution of $\mathcal{T}\|\mathcal{T}'$ satisfies $\mathcal{A}$ for given $\mathcal{T}, \mathcal{T}'$, and $\mathcal{A}$ [30].

To prove the hardness we reduce from the problem of deciding the truth of a formula of the form $\exists x \forall y \Phi(x, y)$, where x and y are vectors of Boolean variables and Φ is a formula in disjunctive normal form [13]. Let $x = (x_1, \ldots, x_k)$, $y = (y_1, \ldots y_k)$ (we assume without loss of generality that x, y have the same number k of variables), and let $\Phi = C_1 \vee C_2 \vee \ldots \vee C_p$, where each term C_i is a conjunction of literals (variables in x or y or their negations). We construct an instance $(\mathcal{A}, n, m)$ of the strong realizability problem such that $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A}, n, m)$ iff $\exists x \forall y \Phi(x, y)$.

The integers n, m are both set to $2p + 1 + k$. The input and output alphabets have size $2p + 1 + 2k$: The input alphabet is $\Sigma_I = \{d_0\} \cup \{d_i, d'_i | 1 \leq i \leq p\} \cup \{y_j, \bar{y}_j | 1 \leq j \leq k\}$, and the output alphabet is $\Sigma_O = \{c_0\} \cup \{c_i, c'_i | 1 \leq i \leq p\} \cup \{x_j, \bar{x}_j | 1 \leq j \leq k\}$.

[4] A similar reduction used in related lower bounds, e.g., to the problem of finding minimal counterexamples in model checking [6] or of finding minimal witnesses to the nonemptiness of NBWs [15].

Recall that the alphabet of the DBW $\mathcal{A}$ is $\Sigma_O \times \Sigma_I$ and it specifies the expected behavior of the system with respect to all input sequences. Thus, $\mathcal{A}$ should not limit the behavior of the environment. Since, however, the interaction between the system and the environment would be of interest only for some behaviors of the environment, we are going to describe how $\mathcal{A}$ prescribe an expected format of interaction for both the system and the environment, and we assume that if, during the interaction, the system deviates from this format then $\mathcal{A}$ moves to rejecting sink, and if the system follows the format but the environment deviates, then $\mathcal{A}$ accepts.

The DBW $\mathcal{A}$ prescribes the expected format in two phases. The goal of Phase 1 is to force the system and the environment to commit to a truth assignments for the variables in x and y, respectively, and to set aside states that output $c_1, c'_1, \ldots, c_p, c'_p$, and $d_1, d'_1, \ldots, d_p, d'_p$, respectively. Phase 1 consists of two stages. During Stage 1, the system commits to values to the variables in x and sets aside states that output $c_1, c'_1, \ldots, c_p, c'_p$. Stage 1 involves $k + 2p$ steps. Recall that we consider 1-computations, where the system starts the interaction. In step $j = 1, \ldots, k$, the system has generate either x_j or $\bar{x}_j$, and then the environment has to generate d_0 (as explained above, if the system generates a different letter, then $\mathcal{A}$ rejects, and if the environment generates a different letter then $\mathcal{A}$ accepts). In the final $2p$ steps of Stage 1, the system has to generate $c_1, c'_1, \ldots, c_p, c'_p$, and the environment has to generate d_0.

Stage 2 of Phase 1 is similar to Stage 1, with the roles of the system and environment switched. In all the $k + 2p$ steps of Stage 2, the system has to generate c_0. In step $j = 1, \ldots, k$, the environment has to generate either y_j or $\bar{y}_j$. In the last $2p$ steps, the environment has to generate $d_1, d'_1, \ldots, d_p, d'_p$.

Before describing how $\mathcal{A}$ forces the above format, note that if the system follows the prescribed format in Phase 1, then it outputs $2p + 1 + k$ different symbols during this phase: c_0, c_i, c'_i for $i = 1, \ldots, p$, and either x_j or $\bar{x}_j$ for $j = 1, \ldots, k$. Since the bound n on its number of states is $2p + 1 + k$, this implies that each state of the system has as output one of the letters above. In particular, this means that for each variable x_j, the system has exactly one corresponding state that outputs x_j or $\bar{x}_j$, which corresponds to assigning value true or false, respectively, to the variable x_j. Similarly, if the environment follows the prescribed format in Phase 1, and then it has exactly $2p + 1 + k$ states, corresponding to the n lettes generated d_0, d_i, d'_i for $i = 1, \ldots, p$, and either y_j or $\bar{y}_j$ for $j = 1, \ldots, k$, and they induce a truth assignment for the y variables.

The first portion of $\mathcal{A}$, corresponding to Phase 1, has one state s_i for each step i and it checks that the system and the environment follow the prescribed format: for $i = 1, \ldots, 2(k + 2p)$, state s_i of $\mathcal{A}$ makes a transition to state s_{i+1} if the symbols of both the system and the environment conform to step i of Phase 1 (for example, in state s_1, the DBW $\mathcal{A}$ expects to read x_1/d_0 or $\bar{x}_1/d_0$); otherwise it makes a transition to the rejecting sink if the system does not conform, and to the accept sink if the system conforms but the environment does not. State $s_{2(k+2p)+1}$, reachable from $s_{2(k+2p)}$ with c_0/d'_p, is the first state of the portion of $\mathcal{A}$ corresponding to Phase 2, described below.

During Phase 2, the DBW $\mathcal{A}$ checks that the assignment to which the system and environment commit in Phase 1 satisfies Φ. Phase 2 consists of p stages, one for each term C_i of Φ. In the stage for C_i, the DBW $\mathcal{A}$ goes over the $2k$ variables and checks whether the assignment for them satisfies C_i. Recall that C_i is a conjunction. When $\mathcal{A}$ detects that the assignment contradicts a requirement imposed by C_i, it moves to the phase for C_{i+1}, or rejects, if $i = p$. When $\mathcal{A}$ concludes that the assignment satisfies C_i, it accepts. In order to simplify the check, we assume that each term C_i is given by a sequence $(x_1, a^i_1), (x_2, a^i_2), \ldots, (x_k, a^i_k), (y_1, b^i_1), (y_2, b^i_2), \ldots, (y_k, b^i_k)$, there a^i_j and b^i_j are flags in $\{-, +, \perp\}$. A flag $-$ indicates that the variable appears negatively in C_i, a flag $+$ indicates it appears positively, and $\perp$ indicates it does not appear in C_i (we assume that no variables appears both positively and negatively). This enables us to assume that all terms are of length exactly $2k$, with the x_j's being ordered before the y_j's.

The portion of $\mathcal{A}$ that checks the term C_i during Phase 2 has $2k+1$ states: $q_{i,1}, \ldots, q_{i,k}, s_{i,1}, \ldots, s_{i,k}$,

and $q_i^{\hookrightarrow}$. At state $q_{i,j}$, it expects to read x_j/d_i or $\bar{x}_j/d_i$, and it checks whether the value of x_j agrees with the one imposed by C_i. If the check is successful (that is, $a_j^i = \bot$ or a_j^i has a polarity that agrees with the value being read), then $\mathcal{A}$ moves to $q_{i,j+1}$ (or to $s_{i,1}$, if $j = k$). If the check is unsuccessful, then $\mathcal{A}$ proceeds to $q_i^{\hookrightarrow}$, from which it starts the stage of Phase 2 that corresponds to the next term. At state $s_{i,j}$, the DBW $\mathcal{A}$ expects to read c_i/y_j or $c_i/\bar{y}_j$, and again, proceeds according to the value of y_j and b_j^i: to $s_{i,j+1}$ if $b_j^i = \bot$ or the polarity agrees with the value being read (or to the accepting sink, if $j = k$), and to $q_i^{\hookrightarrow}$ (or to the rejecting sink, if $i = p$) otherwise. Finally, in state $q_i^{\hookrightarrow}$ the DBW $\mathcal{A}$ expects to read c_i'/d_i', with which it moves to $q_{i+1,1}$ (or, if $i = p$, then $\mathcal{A}$ moves to a rejecting sink). As in Phase 1, whenever the letter is not of the expected type, $\mathcal{A}$ moves to the accepting or rejecting sink, depending on whether the violation is due to the environment or system, respectively.

Intuitively, d_0 and c_0 are generated by the environment and the system, respectively, only in Phase 1. The input d_i is generated by the environment in Phase 2 when it prompts the system to generate the assignment for the x variables during the check of the term C_i. If a contradiction to C_i is detected, the system generates c_i' and the environment responds with d_i', indicating that the check of C_i has not been successful and a check of C_{i+1} is in order. The inputs y_j and $\bar{y}_j$ are generated by the environment when the system prompts it to generate the assignment for the y variables. The system does so by generating c_i. Again, if a contradiction is detected, the system generates c_i' and the environment responds with d_i'.

We prove that $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A}, n, m)$ iff $\exists x \forall y \Phi(x, y)$. Assume first that $\exists x \forall y \Phi(x, y)$. Thus, there is a truth assignment σ to the x variables such that for every truth assignment for the y variables, there is at least one term of Φ that has all its literals true. We construct a system transducer $\mathcal{T}$ according to the assignment σ: the transducer has $n = 2p + 1 + k$ states that output c_0, c_i, c_i' for $i = 1, \ldots, p$, and whichever literal x_j or $\bar{x}_j$ is true according to the assignment σ, for $j = 1, \ldots, k$. For simplicity, let us denote the first $2p + 1$ states by their output symbols and a state in which x_i or $\bar{x}_i$ is generated by $\tilde{x}_i$. Also, let us refer to input letters y_j or $\bar{y}_j$ by $\tilde{y}_j$. The transitions of $\mathcal{T}$ are defined so that the system always follows the prescribed format. Thus, on input d_0 there is a sequence of transitions as required in Stage 1 of Phase 1: from state $\tilde{x}_i$ to $\tilde{x}_{i+1}$ for $i = 1, \ldots, k-1$, from $\tilde{x}_k$ to c_1, then to c_1', and so forth, and finally from c_p' to c_0. To conform to Stage 2 of Phase 1, state c_0 has a self-loop on all inputs, except for d_p' on which it transitions to state $\tilde{x}_1$, ready to start Phase 2.

For Phase 2, the transitions are also defined according to the prescribed format: From $\tilde{x_j}$, on d_i, if $a_i^j = \bot$ or the polarity of x_j agrees with a_i^j, the transition is to $\tilde{x}_{j+1}$, or to c_i, if $j = k$. If the polarity of x_j does not agree with a_i^j, the transition is to c_i'. From c_i, on $\tilde{y}_j$, if $b_i^j = \bot$ or the polarity of y_j agrees with b_i^j, then stay in c_i. If the polarity of y_j does not agree with b_i^j, go to c_i'. Finally, from the state c_i', for $i < p$, on input d_i', the transition is to $\tilde{x}_1$, ready to start the check for C_{i+1}. In all other cases (which correspond to the environment deviating from the prescribed format, and hence to $\mathcal{A}$ reaching the accepting sink), the transition is arbitrary.

By the definition of $\mathcal{T}$, for every transducer $\mathcal{T}'$ for the environment, the system follows the prescribed format in the execution of $\mathcal{T} \| \mathcal{T}'$ as long as the environment does. We claim that for every environment transducer $\mathcal{T}'$ with at most n states, the execution of $\mathcal{T} \| \mathcal{T}'$ is accepted by $\mathcal{A}$. If the environment deviates at some point from the prescribed format, then $\mathcal{A}$ accepts, and we are done. If the environment does not deviate, then the system does not deviate either and the execution of $\mathcal{T} \| \mathcal{T}'$ follows the prescribed format. From Phase 1 we know that $\mathcal{T}'$ has n states and induces a truth assignment τ to the y variables. From the choice of the assignment σ to the variables x, we know that there is a term of Φ all of whose literals are true under the assignments σ, τ; let C_i be the first such term. Since the execution does not deviate from the prescribed format and all the literals of C_i are true, the automaton $\mathcal{A}$ would reach Stage i of Phase 2 (if $i > 1$, then all earlier stages $l < i$ would end with c_l'/d_l' being read), would traverse successfully through all the states $q_{i,1}, \ldots, q_{i,k}, s_{i,1}, \ldots, s_{i,k}$ of the automaton, and would then move to an accepting sink from $s_{i,k}$, so we are done.

For the other direction, assume that $\exists x \forall y \Phi(x, y)$ is false. Consider a transducer $\mathcal{T}$ for the system with n states. We argue that there is a transducer $\mathcal{T}'$ with n states for the environment such that $\mathcal{A}$ rejects the 1-execution of $\mathcal{T} \| \mathcal{T}'$.

Because of Phase 1, the transducer $\mathcal{T}$ must have $2p + 1$ states that output $d_0, d_1, d'_1, \ldots, d_p, d'_p$ and k more states that output x_j or $\bar{x}_j$ for $j = 1, \ldots, k$, corresponding to a truth assignment σ for the x variables. If this is not the case, then clearly there is an environment transducer $\mathcal{T}'$ with n states such that the system deviates in Phase 1 of the execution of $\mathcal{T} \| \mathcal{T}'$ and $\mathcal{A}$ rejects. Since $\exists x \forall y \Phi(x, y)$ is false, there is an assignment τ for the y variables such that every term of Φ contains at least one false literal. From the assignment τ we can construct a transducer $\mathcal{T}'$ for the environment in a similar manner as we did above for the system transducer, which induces the truth assignment τ for the y variables and which conforms to the prescribed format as long as the system transducer also conforms. In particular, whenever $\mathcal{T}$ generates $\tilde{x}_i$, the transducer $\mathcal{T}'$ can consult its current state, which is one of $d'_p, d'_1, \ldots, d'_{p-1}$ in order to know which term is being checked now.

Consider the execution of $\mathcal{T} \| \mathcal{T}'$. If the system deviates first, then $\mathcal{A}$ rejects, so assume that the system conforms, and hence the environment also conforms to the prescribed format. Since every term C_i of Φ contains a false literal according to the truth assignments σ, τ, in Phase 2 of the execution of $\mathcal{T} \| \mathcal{T}'$, the automaton $\mathcal{A}$ goes through the states $q_i^{\hookrightarrow}$ for all $1 = 1 \ldots p$, and proceeds to the rejecting sink in $q_p^{\hookrightarrow}$, and we are done. ◀

We can now turn to the problem of synthesis with bounded environments. We first need some definitions. For a word $w/y \in (\Sigma_I \times \Sigma_O)^*$ and $k \geq 1$, we say that w/y is *k-generable* if there is a Σ_I/Σ_O-transducer $\mathcal{T}$ with at most k states such that $\mathcal{T}(w) = w/y$. Let $L_k \subseteq (\Sigma_I \times \Sigma_O)^*$ be the set of k-generable words.

▶ **Lemma 11.** *There is an NFW with $k^{O(k)}$ states that recognizes L_k.*

Proof. The NFW guesses a Σ_I/Σ_O-transducer $\mathcal{T}$ with at most k states and then simulates it, checking that the y track is indeed the output of the w track. The number of Σ_I/Σ_O-transducers $\mathcal{T}$ with exactly k states is $k|\Sigma_O|^k k^{k|\Sigma_I|}$, and the number of transducers with at most k states, is no more than k times this expression. ◀

▶ **Corollary 12.** *There is a DFW with $2^{k^{O(k)}}$ states that recognizes L_k.*

We can now solve the strong realizability problem for bounded environments.

▶ **Theorem 13.** *Deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}, \infty, k)$, for $b \in \{1, 2\}$ is in 2EXPTIME.*

Proof. We prove the claim for $b = 1$. The argument for $b = 2$ is analogous.

We recall first how ones decides $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A}, \infty, \infty)$. The key idea is to consider the following game between two players, called System and Environment. In each round, System first chooses a letter in Σ_I and Environment then chooses a letter in Σ_O. System wins if the play, which is the infinite word in $(\Sigma_I \times \Sigma_O)^\omega$ that results from the choices of System and Environment is accepted by $\mathcal{A}$. Checking if System wins the game is equivalent to testing nonemptiness of Büchi tree automata and can be done in quadratic time [32]. Furthermore, if System wins the game, then there is a strategy that depends only on the states of $\mathcal{A}$. Thus, if $\mathcal{A} = \langle \Sigma_I \times \Sigma_O, Q, q_0, \rho, F \rangle$, then the nonemptiness algorithm yields a function $L : Q \to \Sigma_O$, which means that $\mathcal{A}$ is strongly realized by the transducer $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \eta, L \rangle$, where $\eta(q, \sigma) = \rho(q, \langle \sigma, L(q) \rangle)$.

In deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}, \infty, k)$, System only has to win against environments with at most k states. Thus, if a prefix of the play is not k-generable then System wins immediately, since no transducer with at most k states can generate such a play. Let $\mathcal{D}_k$ be a DFW that accepts exactly all words that are not k-generable. By Corollary 12, such a DFW with $2^{2^{O(k)}}$ states exists.

Then, $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$ holds iff $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}\times\mathcal{D}_k,\infty,\infty)$, where $\mathcal{A}\times\mathcal{D}_k$ is the product of $\mathcal{A}$ and $\mathcal{D}_k$, which accepts a word $w\in(\Sigma_I\times\Sigma_O)^\omega$ if w is accepted by $\mathcal{A}$ or is not k-generable. It follows that if $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$, then $\mathcal{A}$ is realized by a transducer who states space is that of $\mathcal{A}\times\mathcal{D}_k$. ◀

Theorem 13 tells us that we can solve strong realizability against bounded transducers, but at a cost that is doubly exponential in k. Thus, on the one hand, we expect more specifications to be realizable when we bound the size of the adversaries, but, on the other hand, deciding such realizability comes at a considerable cost. The question is whether this cost is unavoidable. To prove this, we would have to show that deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$ is 2EXPTIME-hard.

OPEN QUESTION: Is deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$ 2EXPTIME-hard?

As in the case of bounded systems, we can show that the strong realizability problem for bounded environment is at least NP-hard. The problem, however, seems to be much harder. As supporting evidence for the hardness of the problem, we show that recognizing k-generable words by an automaton requires doubly-exponential size.

For a word $x=x_1,\ldots x_n\in\Sigma_I^*$, a *combination lock* transducer for x is $\mathcal{T}_x=\langle\Sigma_I,\Sigma_O,\{0,\ldots,n\},0,\tau\rangle$, where $\Sigma_O=\{0,1\}$, and for all $0\leq j<n$, we have $\rho(j,x_{j+1})=j+1$ and $\rho(j,\sigma)=0$, for all $\sigma\neq x_{j+1}$. Also, $\rho(n,\sigma)=n$, for all $\sigma\in\Sigma_I$. Finally, $\tau(j)=0$ for all $0\leq j<n$ and $\tau(n)=1$. Thus, $\mathcal{T}_x$ outputs 0 in all states but n. It can read x from its initial state, in which case it reaches the state n, where it outputs 1. When a violation of x is detected, $\mathcal{T}_x$ goes back to its initial state.

▶ **Theorem 14.** *A DFW that recognizes L_k has at least 2^{2^k} states.*

Proof. Recall that two words $x_1,x_2\in(\Sigma_I\times\Sigma_O)^*$ are L_k-equivalent, in the Myhill-Nerode sense, iff for all $z\in(\Sigma_I\times\Sigma_O)^*$, we have that $x_1\cdot z\in L_k$ iff $x_2\cdot z\in L_k$. The number of states in a minimal DFW for L_k is the number of equivalence classes of the L_k-equivalence relation.

Let $\Sigma_I=\{a,b,\#\}$ and $\Sigma_O=\{0,1\}$. For a word $x\in(a+b)^k$, let $\mathcal{T}_x$ be the transducer obtained from the combination lock $\{a,b\}/\{0,1\}$-transducer for x by adding $\#$-transitions from all states to the initial state. Thus, $\#$ is an input reset symbol that resets to the initial state from all states. For every subset P of $(a+b)^k$, let $w(P)$ be the Σ_I/Σ_O word in which the input part consists of the words in P in some order, say lexicographic, with each word preceded by a reset, and the output part is all 0. We claim that all the words $w(P)$, for different subsets P, are not L_k-equivalent. Let P and Q be two different subsets. Assume, without loss of generality, that $x\in Q\setminus P$.

Note that the transducer T_x is strongly connected, and it is easy to see also that it is minimized. Let w be a checking sequence for $\mathcal{T}_x$, and let $\mathcal{T}_x(w)=w/y$. That is, the only k-state transducer that can generate w/y (including k-transducers that are not combination locks) is $\mathcal{T}_x$ starting from some state. If we append to $w(P)$ the word $\#/0\cdot w/y$, then the resulting word is in L_k, as $\mathcal{T}_x$ generates it. On the other hand, if we append the word $\#/0\cdot w/y$ to $w(Q)$, then the resulting word is not in L_k. Indeed, because of the w/y portion, the only k-transducer that could possibly generate the resulting word is $\mathcal{T}_x$, but since $x\in Q$, the transducer $\mathcal{T}_x$ cannot generate $w(Q)$, as it would output 1 after reading $\#x$. [5] ◀

[5] A small variant of the argument holds also for binary input alphabet, i.e., without the additional reset symbol. Restrict to combination locks where the first and the last symbol of the combination word is a, and replace the reset symbol by b^k. Then, b^k acts like a reset for these machines.

5.1 Absolute liveness properties

A language $L \subseteq \Sigma^\omega$ is an *absolute liveness* language if $L = \Sigma^* \cdot L$. Thus, L is insensitive to additions of prefixes [28]. [6] A specification ψ is of an absolute liveness property the set of computations that satisfy ψ constitute an absolute liveness language. Equivalently, for every computation π, we have that $\pi \models \psi$ iff $\pi \models F\psi$ [28]. In this section we show that for absolute liveness properties, weak and strong realizability against bounded environments coincide, and the complexity of deciding realizability is only exponential in the bound. Intuitively, it follows from the fact that the system can take its time to learn the environment with which it interacts, and then follow a strategy against that environment.

We formalize this intuition by means of the so-called *machine identification problem*: Given $k \geq 1$, we say that a word w is a *k-identifier* for every two O/I-k-transducers $\mathcal{T}_1', \mathcal{T}_2'$, if the two transducers produce the same output sequence y in response to w, and if s_1, s_2 are their states respectively after processing w, then $\mathcal{T}_1'/s_1 \equiv \mathcal{T}_2'/s_2$. In other words, observing the response of the environment to w, identifies uniquely up to equivalence the part of the environment transducer that is reachable from the final state *after* w. The machine identification problem was formulated and solved by Moore in his classical paper [21]. It is shown there that for every $k \geq 1$, there is a k-identifier of length exponential in k, and it can be constructed in exponential time. (The word w is essentially a homing sequence of the disjoint union of all k-transducers.)

▶ **Theorem 15.** *Let ψ be an absolute liveness specification. Then*

1. *$w\_real_{I,O,1}(\psi, \infty, k)$ iff $s\_real_{I,O,1}(\psi, \infty, k)$.*
2. *If ψ is given as a DBW, we can decide $s\_real_{I,O,1}(\psi, \infty, k)$ in time polynomial in ψ and exponential in k.*

Proof. Clearly $s\_real_{I,O,1}(\psi, \infty, k)$ implies $w\_real_{I,O,1}(\psi, \infty, k)$. For the other direction, assume that $w\_real_{I,O,1}(\psi, \infty, k)$ holds. Then, for each O/I-k-transducer $\mathcal{T}'$ there is an I/O-transducer $\mathcal{T}$ that guarantees that ψ holds. We need, however, one I/O-transducer $\mathcal{T}$ that can guarantee ψ against all O/I-k-transducers. What $\mathcal{T}$ can do is first output a k-identifier sequence w. After observing the response y of the k-transducer $\mathcal{T}'$ of the environment, we can construct a transducer $\mathcal{T}''$ with at most k states that is equivalent to $\mathcal{T}'/s$ where s is the current state of $\mathcal{T}'$ after w. Then, using weak realizability, $\mathcal{T}$ can simulate the I/O-transducer that wins against $\mathcal{T}''$. This proves the first claim.

The second claim follows from the fact that, for every given k-transducer $\mathcal{T}'$ for the environment, we can determine in polynomial time whether there is a system that satisfies a DBW specification ψ with environment $\mathcal{T}'$, hence $w\_real_{I,O,1}(\psi, \infty, k)$ (and $s\_real_{I,O,1}(\psi, \infty, k)$) can be decided in time polynomial in ψ and exponential in k. If $s\_real_{I,O,1}(\psi, \infty, k)$ holds, then a transducer $\mathcal{T}$ for the system that realizes ψ can be constructed also in time polynomial in ψ and exponential in k, as explained above. ◀

References

1 B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.

2 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

3 J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

[6] Note that the notion of absolute liveness is stronger than the notion of liveness: L is a liveness language of if for every finite word $x \in \Sigma^*$, there is an infinite word $y \in \Sigma^\omega$ such that $x \cdot y \in L$ [1].

4 K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proc. 19th Int. Conf. on Concurrency Theory*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008.

5 A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.

6 E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32st Design Automation Conf.*, pages 427–432. IEEE Computer Society, 1995.

7 R. Ehlers. Symbolic bounded synthesis. In *Proc. 22nd Int. Conf. on Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2010.

8 E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 79–87. Springer, 1985.

9 M. Faella, M. Napoli, and M. Parente. Graded alternating-time temporal logic. In *Proc. 16th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*, volume 6355 of *Lecture Notes in Computer Science*, pages 192–211. Springer, 2010.

10 E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643, pages 263–277, 2009.

11 C. Finucane, G. Jing, and H. Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 1988 – 1993, 2010.

12 D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010.

13 M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. Freeman and Co., 1979.

14 A. Harding, M. Ryan, and P. Schobbens. A new algorithm for strategy synthesis in LTL games. In *Proc. 11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2005.

15 O. Kupferman and S. Sheinvald-Faragy. Finding shortest witnesses to the nonemptiness of automata on infinite words. In *Proc. 17th Int. Conf. on Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 492–508. Springer, 2006.

16 O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

17 O. Kupferman and M.Y. Vardi. Synthesis of trigger properties. In *Proc. 16th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*, volume 6355 of *Lecture Notes in Computer Science*, pages 312–331. Springer, 2010.

18 D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. *Proc. IEEE*, 84:1089–1123, 1996.

19 D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.

20 A. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

21 E.F. Moore. Gedanken-experiments on sequential machines. In *Automata studies*, volume 34 of *Ann. Math. Studies*, pages 129–153. Princeton Univ. Press, 1956.

22 C.H. Papadimitriou and M. Yannakakis. On complexity as bounded rationality (extended abstract). In *Proc. 26th ACM Symp. on Theory of Computing*, pages 726–733, 1994.

23 N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive(1) designs. In *Proc. 7th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 3855 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2006.

24 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.

25 R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.

26 A. Rubinstein. Finite automata play the repeated prisoner's dilemma. *J. Economic Theory*, 39(1):83 – 96, 1986.

27 S. Schewe and B. Finkbeiner. Bounded synthesis. In *5th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2007.

28 A.P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.

29 A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32:733–749, 1985.

30 M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.

31 M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

32 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.

33 G. Wang, A. Mishchenko, R. Brayton, and A. Sangiovanni-Vincentelli. Synthesizing FSMs according to co-Büchi properties. Technical report, UC Berkeley, 2005.