Adapting to the Behavior of Environments with Bounded Memory*

Dhananjay Raju
The University of Texas at Austin, USA
draju@cs.utexas.edu

Rüdiger Ehlers
Clausthal University of Technology, Germany
ruediger.ehlers@tu-clausthal.de

Ufuk Topcu
The University of Texas at Austin, USA
utopcu@utexas.edu

We study the problem of synthesizing implementations from temporal logic specifications that need to work correctly in all environments that can be represented as transducers with a limited number of states. This problem was originally defined and studied by Kupferman, Lustig, Vardi, and Yannakakis. They provide NP and 2-EXPTIME lower and upper bounds (respectively) for the complexity of this problem, in the size of the transducer. We tighten the gap by providing a PSPACE lower bound, thereby showing that algorithms for solving this problem are unlikely to scale to large environment sizes. This result is somewhat unfortunate as solving this problem enables tackling some high-level control problems in which an agent has to infer the environment behavior from observations. To address this observation, we study a modified synthesis problem in which the synthesized controller must gather information about the environment's behavior *safely*. We show that the problem of determining whether the behavior of such an environment can be safely learned is only co-NP-complete. Furthermore, in such scenarios, the behavior of the environment can be learned using a Turing machine that requires at most polynomial space in the size of the environment's transducer.

1 Introduction

Reactive synthesis is the process of automatically computing correct (by construction) implementations of systems from their formal specifications [3, 11, 17]. A synthesized system is guaranteed to satisfy its specification along all of its executions, regardless of how the environment behaves. In this way, synthesis is much stronger than *planning* (in deterministic domains), i.e., the process of finding one execution of a system satisfying the specification from its current state [20], as synthesis includes planning for all possible behaviors of the environment. However, there are many cases in which reactive synthesis fails because there is no system that satisfies the specification against *all* environment behaviors. This is for instance the case in scenarios in which the environment can block the system from achieving its objectives [15]. An example for such a case is depicted in Figure 1, where a human and a robot share a workspace.

The robot controller to be synthesized has the task of evading the human while at the same time recharging whenever necessary by visiting a recharging station and staying there for two time steps in a row. After formalizing the scenario in the form of a specification, a reactive synthesis tool will conclude that there is no implementation, as the human can always move to the same lane as the robot. This human behavior requires the robot to back up as it is its task to evade. In this way, the robot can

*This material is based upon work supported by ARL ACC-APG-RTP W911NF1920333, AFRL FA9550-19-1-0169, DARPA D19AP00004 and NSF 1652113.

D. Bresolin and P. Ganty (Eds.): 12th International Symposium
on Games, Automata, Logics, and Formal Verification (GandALF 2021)
EPTCS 346, 2021, pp. 52–66, doi:10.4204/EPTCS.346.4

© Dhananjay Raju, Rüdiger Ehlers & Ufuk Topcu
This work is licensed under the
Creative Commons Attribution License.

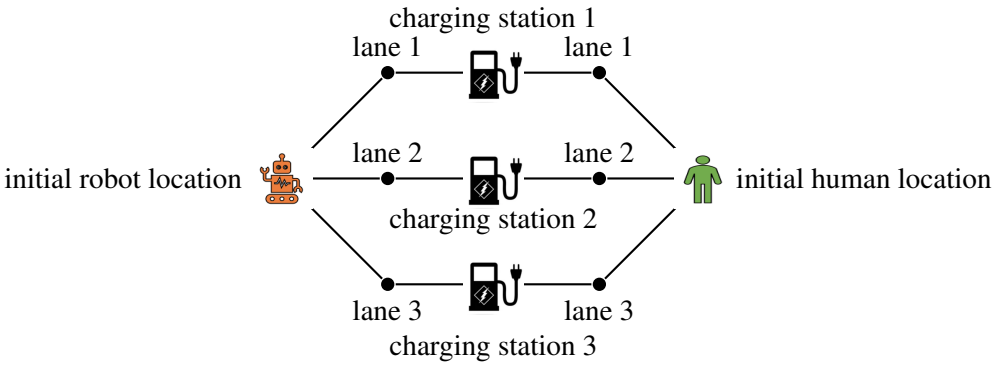

Figure 1: A robot and a human are operating in the same environment. They take turns to move to an adjacent location. The objective of the robotic agent is to use a charging station without colliding with the human. The objective of the human is unknown to the robot.

never use a recharging station. While the answer that there is no controller is correct, a human engineer would typically write a controller by hand for this scenario that manages to recharge correctly whenever the human at least occasionally stays clear from the robot, for instance to pursue its own goals in the workspace.

To weaken the overly strong requirement that the synthesized controller has to always operate correctly no matter how the environment behaves, a common approach is to make specific *assumptions* that restrict the environment's behavior. The synthesized system then has to operate correctly only in environments that satisfy these assumptions. However, this approach creates a new issue that the synthesized implementations are incentivized to work against the satisfaction of the assumptions (which can be partially mitigated [4, 17]). Alternatively, the interaction between environment and system can be viewed from the perspective of strategic games, in which some form of stable equilibrium between the strategies of the environment and system players is computed such that none of the two players are incentivized to deviate [13, 6]. Both approaches require information about the environment's goals to perform this strategic reasoning. Such information about the environment in which the system to be synthesized is supposed to operate in is unfortunately not always available.

In the example from Figure 1, we are seeking a controller that avoids collisions with a human without knowing the human's intention in the shared workspace. A controller can do so by *observing* the behavior of the human and adapting its control policy in a way that the human is avoided. Since environments can behave arbitrarily within their limits (in synthesis), they can also change their behavior arbitrarily and hence past observed behavior is useless in the setting of classical reactive synthesis. This observation leads to the question if an alternative definition for the synthesis problem exists that would enable us to perform formal synthesis of a correct-by-construction controller in unknown environments. Some quantification about the environment's capabilities is necessary to make solving this problem useful, as otherwise the environment can behave fully antagonistically as in classical reactive synthesis. At the same time, we are seeking for a controller that always works correctly against simple behaviors of the environment. This requirement can be formalized by starting from the observation that the size of the memory of a *transducer* encoding the environment's behavior can be used as an abstract notion of the environment's behavioral complexity [19]. 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 crypto-system with respect to attackers with bounded computational power [5]. A symbolic

synthesis procedure for bounded synthesis of lasso-precise implementations based on quantified Boolean formula solving has been provided in [8].

In the scenario from Figure 1, simple behavior of the human (environment) would enable the robot to perform its task, while very complex behavior such as blocking the robot by always moving to the same lane as the robot does not. The human needs to use a state of memory for moving to a particular charging lane. Additionally, it needs one memory state to come back to the initial location. Say the human is using a transducer with three states, then the human can block the robot from charging in at most two lanes. However, with four states, the human can successfully block the robot from charging. The *synthesis under bounded environments* problem has originally been defined by Kupferman, Lustig, Vardi and Yannakakis [16], who also provide an algorithm for this synthesis problem that has a time complexity that is doubly-exponential in the number of states of finite-state machines (also known as *transducers*) representing the environment. However, the lower bound on the complexity that they give is only NP, leaving a hope that this problem can be solved for scenarios of practical relevance, e.g., by employing a satisfiability (SAT) solver.

In this paper, we tighten the gap between the upper and lower complexity bounds of synthesis under bounded environments and provide a new PSPACE lower bound, shattering the hope that the synthesis problem for bounded environments has a (relatively) low complexity. Our proof is based on the observation that in order to solve the problem, the synthesis algorithm needs to distill the safe ways of "probing" the environment in order to obtain information about its behavior, which causes the high complexity. We then prove that if the synthesized controller can observe the environment in a *safe way*, the problem is simpler. This applies, for instance, to the robotics scenario from Figure 1. The robot can always move safely without restricting future behavior due to its past actions. However, for eventually recharging, the robot needs to use the observations made.

To precisely capture such scenarios, we strengthen the original problem formulation by Kupferman, Lustig, Vardi and Yannakakis and introduce the notion of *k-transducer liveness*. A synthesis problem instance is k-transducer live if from every prefix behavior that is compatible with at least one transducer of size k (for the environment), there is a way for the system to continue operating such that it can eventually satisfy its objectives. Computing a controller for the system requires that no "probing" can make the system *irrecoverable*, i.e., get the system into a situation from which it cannot satisfy its specification due to its prior actions. The example in Figure 1 is 3-transducer live, i.e., when the human corresponds to a transducer with three states, the robot can move so as to figure out which two (out of three) charging stations the human may block. The significance of this new notion is that focusing on k-transducer liveness reduces the complexity of the synthesis problem to co-NP-completeness. With the new definition, we are condensing the problem of finding out if it is possible for the system to gradually adapt to the environment's behavior to a more manageable complexity class. While the complexity is still beyond polynomial time, we can employ efficient SAT solvers after encoding the specification to a synthesis game to determine if some simple environment behavior can block the synthesized system from satisfying the specification.

The synthesized controllers in our approach iterate through possible transducers for the environment behavior and use own behavior adapted to a particular environment transducer until the environment is found to react inconsistently with the supposed transducer. Whenever this is found to be the case, the controller switches to the next possible own behavior. Such a strategy can be implemented on a Turing machine that uses at most polynomial space (in the size of the environment's transducer). We leave the problem of computing controller implementations that adapt to the environment as quickly as possible for future work.

In the context of learning the behavior of a bounded environment, the synthesis for *absolute liveness*

properties – properties that are insensitive to additions of prefixes, was shown to be contained in EXPTIME [16]. There are games that are k-transducer live but not absolutely live. For example, in Figure 1, the game is not absolutely live but is 3-transducer live. Additionally, if the game is absolutely live and is winning for the system player against k-transducers, then the game is also k-transducer live. Thus, absolute liveness is a stronger assumption when compared to k-transducer liveness for the purpose of safely probing bounded environments.

The paper is structured as follows. In Section 2, we provide preliminaries that define two-player games used for reactive synthesis, transducers and transducer languages. In Section 3, we prove a PSPACE lower bound of synthesis for bounded environments by encoding a quantified Boolean formula in a Büchi game against a k-transducer environment. In Section 4, we introduce the notion of k-transducer liveness. We show that identifying whether a game is k-transducer live is co-NP-complete. The co-NP-containment proof is constructive and shows how to compute a strategy to win such games. Lastly, we conclude in Section 5.

2 Preliminaries

Let Σ and Γ be finite alphabets. Furthermore, let $\mathsf{A} = \Sigma\Gamma$.

Definition 1 (Parity Game)**.** *A game G between two players P_1 and P_2 is a tuple $\langle V, \Sigma, \Gamma, E, \iota, F\rangle$, where*

- *$V = V_1 \uplus V_2$ is the set of vertices (or positions). V_1 is the set of P_1 vertices and V_2 is set of P_2 vertices.*
- *Σ and Γ are the action sets of P_1 and P_2, respectively.*
- *$E : (\{V_1 \times \Sigma\} \cup \{V_2 \times \Gamma\}) \to V$ is the transition function, where*

$$E(u,a) = E_1(u,a), \text{ if } u \in V_1 \text{ and } a \in \Sigma \text{ and}$$
$$E(v,b) = E_2(v,b), \text{ if } v \in V_2 \text{ and } b \in \Gamma.$$

  Here, $E_1 : V_1 \times \Sigma \to V_2$ and $E_2 : V_2 \times \Gamma \to V_1$ are functions corresponding to transitions from P_1 and P_2 vertices, respectively.
- *$\iota \in V_1$ is the initial vertex.*
- *$F : V \to \mathbb{N}$ is a coloring function.*

A *play* ρ is a (possibly infinite) sequence $u_0 v_0 u_1 v_1 \ldots$ of vertices such that $u_0 = \iota$ and there is a sequence of actions $w = a_0 b_0 a_1 b_1 \ldots$ such that $E(u_i, a_i) = v_i$ and $E(v_i, b_i) = u_{i+1}$ ($i \in \mathbb{N}$). Moreover, we say that the play ρ is *generated* by the word w. A play $\rho = u_0 v_0 u_1 v_1 \ldots$ is winning for player 2 if and only if the largest number occurring infinitely often in the sequence $F(u_0)F(v_0)F(u_1)\ldots$ is even. A *Büchi* game is a variant of the parity game such that $F : V \to \{1,2\}$. In a Büchi game, a play ρ is winning for P_2 if some vertex v such that $F(v) = 2$ is repeated infinitely often. Throughout this paper, all plays that are not winning for P_2 are winning for P_1. Lastly, *reachability* games are a variant of Büchi games. For them, a play ρ is winning for P_2 if it eventually reaches a vertex v such that $F(v) = 2$.

A strategy σ for a player $P \in \{P_1, P_2\}$ maps every finite prefix sequence of actions $w \in \mathsf{A}^* \cup \mathsf{A}^*\Sigma$ ending with a action for the respective other player to a next action of P (where for P_1, σ also maps the empty word to an element of Σ). A word $w \in \mathsf{A}^* \cup \mathsf{A}^\omega$ is said to *agree* with a strategy σ for P_1 if the play ρ generated by w agrees with σ. A strategy σ for player P is said to be winning if every play ρ (starting at ι) that agrees with σ is winning for player P.

We model finite-state reactive systems with inputs in Γ and outputs in Σ by *transducers*. We use these transducers to model bounded memory environments.

Definition 2 (Transducer)**.** *A finite transducer T is a tuple $\langle \Sigma, \Gamma, M, s, L, \eta \rangle$ that consists of the following components:*

- *M is a finite set of states,*
- *Σ is a set of alphabet symbols, called the output alphabet,*
- *Γ is a set of alphabet symbols, called the input alphabet,*
- *$s \in M$ is the initial state,*
- *$\eta : M \times \Gamma \to M$ is a function, called the transition function and*
- *$L : M \to \Sigma$ is a function, called the labeling function.*

We extend η to words in Γ^* in the straight-forward way. Thus, $\eta : \Gamma^* \to M$ is such that $\eta(\varepsilon) = s$ and for $x \in \Gamma^*$ and $i \in \Gamma$, $\eta(x \cdot i) = \eta(\eta(x), i)$. We define the labeling function on words in Γ^* , $\widehat{L} : \Gamma^* \to \Sigma$, as $\widehat{L} = L \circ \eta$.

Each transducer T induces a *strategy* $f_T : \Gamma^* \to \Sigma$, where for all $w = a_0 b_0 a_1 b_1 \dots a_n b_n \in (\Sigma \times \Gamma)^*$, we have that $f_T(w) = \widehat{L}(b_0 b_1 \dots b_n)$. Thus, $f_T(w)$ is the action that T outputs after reading the P_2 actions in w. A transducer with k states is called a *k-transducer*. Furthermore, a strategy induced by a k-transducer is called a *k-transducer strategy*.

In the subsequent sections, we analyze the behavior of P_1 when it is restricted to using k-transducer strategies. For this purpose, we define compatibility of plays with respect to some k-transducer strategy for P_1 as follows.

Definition 3 (Agreement with a k-transducer)**.** *A word $w = a_0 b_0 a_1 b_1 \dots \in \mathsf{A}^* \cup \mathsf{A}^*\Sigma \cup A^\omega$ is said to agree with a k-transducer $T = \langle \Sigma, \Gamma, M, s, L, \eta \rangle$ if for every prefix $a_0 b_0 \dots a_n b_n a_{n+1}$ of w, we have that $\widehat{L}(b_0 b_1 \dots b_n) = a_{n+1}$.*

We define A_k^* (A_k^ω) to be the set of words $w \in \mathsf{A}^*(\mathsf{A}^\omega)$ that agree with some k-transducer for P_1.

Definition 4 (k-transducer language)**.** *We define the k-transducer language for a reachability, Büchi, or parity game G, denoted by $\mathscr{L}_k(G)$, to be the set of words A^ω that agree with some k-transducer T for P_1 and for which the play generated by w is winning for P_2.*

Games of infinite duration, as in Definition 1, are a conceptual model to reduce *reactive synthesis* to the task of finding out for a given specification in some temporal logic such as linear temporal logic (LTL) whether there exists a transducer whose executions all satisfy the given specification, without the possibility to control the input to the transducer [3]. The specification is translated to an automaton over infinite words, which is in turn translated to a game of infinite duration such that the implementations that satisfy the specification are exactly the *system player* strategies in such games. Games with this property are also called *synthesis games* for the respective specification. We will use Büchi games for hardness proofs and parity games for complexity class containment proofs in this paper. The former are a special case of parity games, so that hardness results carry over to the parity game case.

Problem Statement 1. *Given a reachability, Büchi, or parity game $G = \langle V, \Sigma, \Gamma, E, \iota, F \rangle$, does the system ($P_2$) have a winning strategy in the game G against an k-transducer environment (P_1)?*

We note that despite stating our results based on a formalization of the reactive synthesis problem using games in this paper, the environment model definition is the same as the one by Kupferman et al. [16]. Therefore, hardness and containment results in the size of the environment transducers are valid for the reactive synthesis in bounded environments problem as well.

3 General Büchi games against bounded adversaries

We address *reactive synthesis for bounded environments* by studying the problem from Problem Statement 1, which reformulates this variant of reactive synthesis in the scope of games. The current lower bound for the complexity of this problem is NP-hard [16]. We improve this lower bound by showing that the problem is at least PSPACE-hard in this section.

To show the PSPACE-hardness of this problem, we encode a quantified Boolean formula (QBF) formula ψ of the form

$$\psi = \forall x_1 \exists y_1 \forall x_2 \exists y_2 \ldots \forall x_k \exists y_k : \ (C_1 \wedge C_2 \wedge \cdots \wedge C_r)$$

into a reachability game G_ψ on a graph of size $O(k \cdot (r+1))$ that is winning for the system player (P_2) if and only if ψ is valid (equivalent to **true**). The environment player (P_1) can use only $(k+1)$-transducer strategies ($k \in \mathbb{N}$), while the system player has no such restrictions. In the above QBF, $C_1, \ldots, C_r$ are the *clauses*, i.e., disjunctions of literals in $\{x_i, \neg x_i, y_i, \neg y_i : 1 \leq i \leq k\}$. We depict the structure of the game for a specific QBF instance in Figure 2.

In the game, P_1 chooses its actions from the set $\{x_i, \neg x_i : 1 \leq i \leq k\} \cup \{e\}$, while P_2 uses $\{y_i, \neg y_i : 1 \leq i \leq k\}$ as action set. Intuitively, P_1 and P_2 make assignments to the x-variables and y-variables, respectively, using their $\neg x_i / x_i$ and $\neg y_i / y_i$ actions. P_1 is trying to satisfy the formula, while P_2 is trying to falsify the formula. Additionally, P_1 has the possibility of playing an *exit* move (e), which ends the assignment process. The game is played in four phases.

1. In the first phase, P_1 demonstrates that it can play the e action.
2. In the second phase, P_1 and P_2 jointly construct an assignment.
3. In the third phase, the satisfaction of each of the clauses with respect to the assignment is checked.
4. In the final phase, the play ends in a *paradise* for one of the players.

Additionally, if in the first two phases, one of the players plays an illegal action (from any state moves that are not shown in the figure), then the play moves to a paradise (for the opponent) immediately.

The most interesting phase of the game is the third one, which is played in r stages, with stage $1 \leq j \leq r$ corresponding to clause C_j. In this phase, the players have to make the assignments for all their variables turn by turn, once for each clause. Thus, for every clause they each have to play k rounds.

For every $1 \leq j \leq r$ and $1 \leq i \leq 2k$ such that i is odd, the vertex $v^{\top}_{i,j}$ represents the fact that P_1 is currently choosing the value of variable $x_{(i+1)/2}$ in the clause C_j and that the variable values chosen so far already satisfy the clause. Similarly, the vertex $v^{\bot}_{i,j}$ represents the fact that P_1 is currently choosing the value of variable $x_{(i+1)/2}$ in the clause C_j and the variable values chosen so far do not already satisfy the clause. From even i, P_2 is choosing the value for variable $y_{i/2}$. A special case is the e action of player P_1, which immediately leads to a paradise for one of the two players. If P_1 plays it during phase 2, then the play reaches a paradise for P_2 and hence, P_2 wins the game. If P_1 plays e during phase 3, then P_1 wins the game (by reaching a paradise).

The transducer for P_1 has $k+1$ states. In every transducer that wins the game (for P_1), the initial state $s = m_0$ is reserved for generating the e action. The other k states are needed to make an assignment to the x-variables during the second phase of the game. Without loss of generality, we denote the transducer state used for assigning to variable x_i by m_i. This is because the actions required by P_1 to make an assignment to the variable x_i are unique for each i, i.e., the other actions (including e) are illegal. Therefore, all the $k+1$ states of the transducer have to be used by P_1 for the game to enter phase 3. Lastly, the only legal actions for P_1 from $v^{\bot}_{1,1}$ are x_1 and $\neg x_1$. In particular, the exit action e is not allowed. Since P_1 is forced to

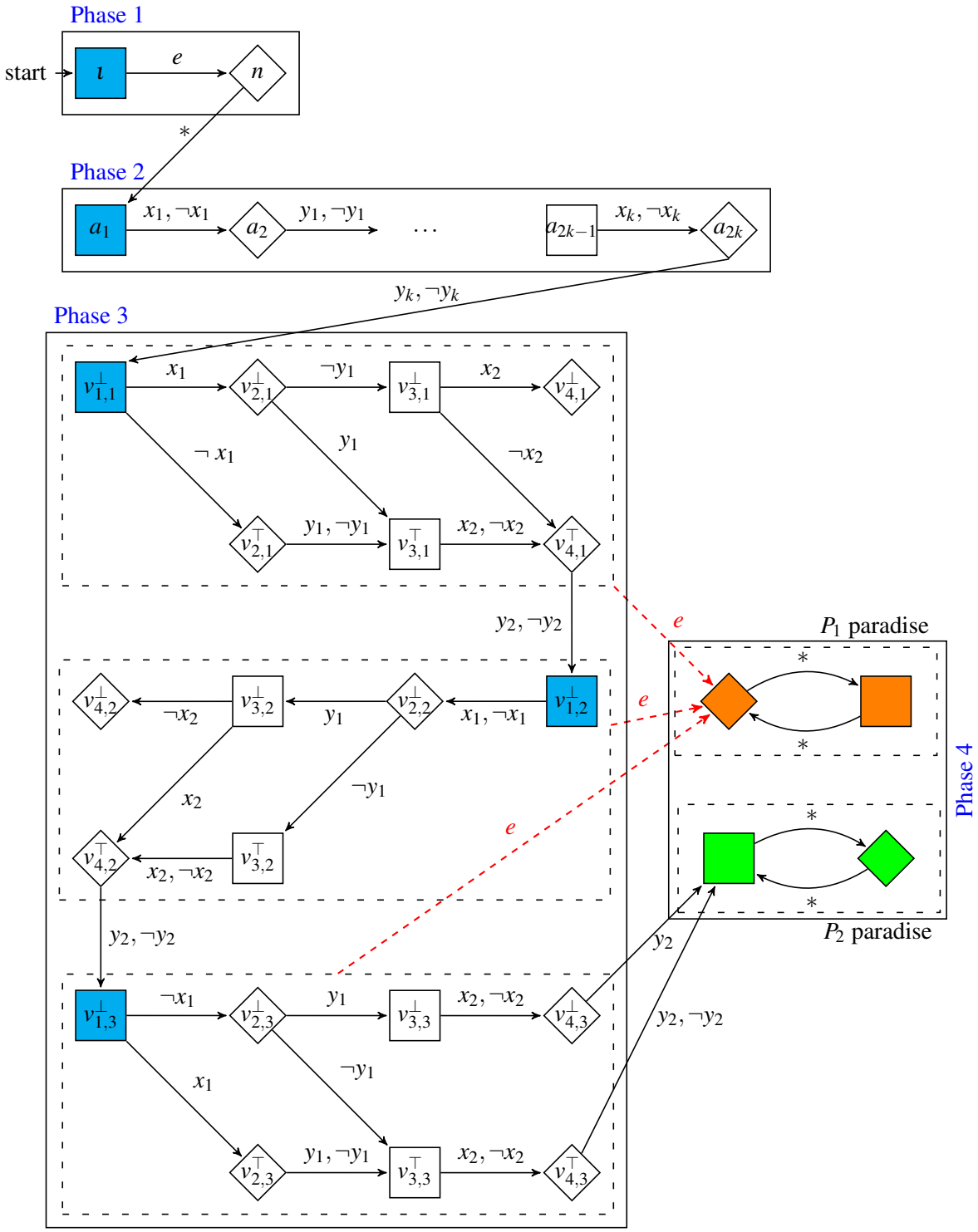

Figure 2: The Reachability game corresponding to the quantified Boolean formula (QBF) $\psi = \forall x_1 \exists y_1 \forall x_2 \exists y_2 : (\neg x_1 \vee y_1 \vee \neg x_2) \wedge (\neg y_1 \vee x_2) \wedge (x_1 \vee \neg y_1 \vee y_2)$. P_1 plays from squared vertices and P_2 plays from diamond-shaped vertices. The *paradises* for the two players are positions from which they (corresponding player) is guaranteed to win a play. The dashed edges labeled by the exit action e correspond to edges that start at every P_1 controlled vertex in phase 3 to the P_1 paradise, except that there is no such transition from $v^{\perp}_{1,1}$. Lastly, if for some action of a player, an outgoing edge is not shown, we assume that this action leads to the *paradise* of its opponent.

use unique memory states to make assignments for the x-variables in phase 1 and the number of memory states $(k+1)$ available for it is exactly equal to the number of different actions required to complete the first two phases, at the start of phase 3, the transducer for P_1 hits the loop $m_1 \to m_2 \cdots \to m_k \to m_1$ on the (memory) states. If it does not hit this loop on its memory states, then it (P_1) would have made an illegal action causing it to lose. However, in the vertices in phase 3, P_1 wins the game if it plays the e action.

At this point, if P_2 continues playing the same assignment (that it had generated in phase 2), the loop on the environment's (P_1) transducer will be continued. However, any deviation from the initial assignment by P_2 could trigger a return to the transducer state m_0 that outputs e, thereby making P_2 lose the game. Hence, P_2 is forced to make the same assignments.

Theorem 1. *The QBF formula ψ is valid if and only if P_2 has a winning strategy against any $(k+1)$-transducer for P_2 in the reachability game constructed from ψ.*

Proof. ($\Rightarrow$) If the formula ψ is valid, then for each $1 \leq i \leq k$, values for y_i can be chosen based only on the values of $x_0, y_0, \ldots, x_{i-1}$ such that after assigning values to all Boolean variables in ψ, all clauses are satisfied. We observe that this gives rise to a strategy for P_2 to win the game built from ψ. When P_2 plays this strategy, at the end of the clause C_j component of the game in phase 3, vertex $v^{\perp}_{1,j+1}$ is reached. For P_1, the only way to then avoid eventually reaching the P_2 paradise is by playing an e move. Since the transducer only has $k+1$ states, after $k+1$ decisions, the transducer has to visit an old state again, in particular the state m_1 giving the value for x_1. Thus, if P_2 repeats the same actions as before, the transducer (for the environment) is then forced to repeat its actions as well, preventing it from ever using the exit action e.
($\Leftarrow$) For the other direction, we show that if the QBF formula is not valid, then for each strategy of P_2, there is a counter-strategy for P_1 that lets P_1 win. The strategy for P_1 is defined as follows:

- In phase 1, P_1 initially plays e from the transducer state m_0.
- In phase 2, P_1 plays the values of x_i from state m_i (for $1 \leq i \leq k$) suitable to eventually falsify some QBF clause for the choices of $x_1, y_1, \ldots, x_{i-1}, y_{i-1}$ made by P_1 and P_2 for $x_1, \ldots, y_{i-1}$ thus far. By the assumption that the QBF is not valid, such a choice is guaranteed to exist for every $1 \leq i \leq k$.
- Finally, in phase 3, P_1 repeats the values of $x_1, \ldots, x_k$ indefinitely unless P_2 chooses values other than $y_1, \ldots, y_k$, in which case it plays the exit move e by transitioning to m_0.

Note that here, the choice of the strategy for P_1 depends on the strategy for P_2. Since we are only asking if there exists a strategy for P_2, this is a valid line of reasoning. By following the above P_1 strategy, the play reaches a P_1 paradise, either due to some clause being violated or due to P_1 using the exit action action e in the third phase (since that indicates that the values of $y_1, \ldots, y_k$ have changed). □

The PSPACE lower bound for the synthesis under bounded environments problem shows that from a computational complexity point of view, the problem is somewhat difficult. In the next section, we provide a method to reduce the complexity by strengthening the synthesis problem. We do so by requiring that the system is able to gather information about the environment's transducer in *safe ways*.

4 k-transducer liveness

In the literature, a linear-time temporal (LTL) property ψ is said to be *live* if and only if for all partial computations α, there is a (possibly infinite) sequence of states β such that $\alpha\beta \models \psi$, i.e., no partial execution is irremediable: it always remains possible for the required 'good thing' to occur in the future [1].

When employing synthesis games that encode the reactive synthesis problem and restricting the environment to behavior implementable as k-transducers, we can rephrase liveness for such environment transducers on the level of games as follows:

Definition 5 (k-transducer liveness)**.** *A game G is k-transducer live if*

$$\forall \alpha \in \mathsf{A}_k^* : \exists \beta \in \mathsf{A}^\omega : \ \alpha\beta \in \mathscr{L}_k(G).$$

For safely observing the behavior of the environment, we require that no finite play that is generated by a k-transducer environment can cause a system failure irrespective of the choice of moves made by the system so far. The above definition of liveness ensures that any finite play agreeing with some k-transducer strategy can always be extended to a play that is winning for the system (P_2). For example, the scenario in Figure 1 is 3-transducer live.

We call a reactive synthesis problem instance *k-transducer live* if the synthesis games encoding the problem instance are k-transducer live. Since k-transducer liveness is given on the level of action sequences, either all synthesis games for a specification have this property, or none of them have. Next, we analyze the complexity of determining if a given game is k-transducer live.

Theorem 2. *Deciding whether a parity game G is k-transducer live is contained in co-NP.*

Proof. We show that it suffices to check for all possible k-transducers separately that P_2 can win from every reachable combination of game position and transducer state. Hence, a co-NP algorithm can non-deterministically guess a transducer T and perform this check. The game is k-transducer live if and only if for all transducers, the answer is "yes". Containment in co-NP follows from this observation.

For each non-deterministically guessed k-transducer $T = \langle \Sigma, \Gamma, M, s, L, \eta \rangle$ and a given game $G = \langle V, \Sigma, \Gamma, E, \iota, F \rangle$, the co-NP algorithm builds a new parity game $\widetilde{G} = \langle \widetilde{V}, \Sigma, \Gamma, \widetilde{E}, \widetilde{\iota}, \widetilde{F} \rangle$ in which P_1's moves are forced to be compatible with the transducer T and the game is played on a graph with vertex set $\widetilde{V} = V \times M$.

Without loss of generality, let $\top \in V$ be a vertex of P_2 in G from which P_2 cannot lose a suffix play. If such a vertex does not exist in the graph, we just add it to G. For every $m \in M$, $u \in V_1$, $v \in V_2$, $a \in \Sigma$, and $b \in \Gamma$, we define:

$$\widetilde{E}((u,m),a) = \begin{cases} (E_1(u,a),m), & \text{if } a = L(m), \\ (\top, m) & \text{otherwise.} \end{cases}$$
$$\widetilde{E}((v,m),b) = (E_2(v,b), \eta(m,b)).$$
$$\widetilde{\iota} = (\iota, s).$$
$$\widetilde{F}((u,m)) = F(u).$$
$$\widetilde{F}((v,m)) = F(v).$$

In the game $\widetilde{G}$, P_1 is restricted to play exactly the strategy induced by T. If P_1 does not follow this strategy, then it loses. It can be tested if P_2 has a strategy against T from every position (v,m). Since P_1's action from every transducer state is fixed, the game becomes a deterministic parity automaton, for which the emptiness of the languages of the automaton's states can be determined in time polynomial in the size of the game [14]. To complete the proof, we now prove the following two sub-claims:

Claim: 1 If for some transducer T, there exists a position (v,m) reachable from $\tilde{\iota}$ in $\widetilde{G}$ that is losing for P_2, then G is not k-transducer live.

Claim: 2 If G is not k-transducer live, then there exists a transducer T such that some position (v,m), reachable from $\widetilde{\iota}$, is losing for P_2 in $\widetilde{G}$.

Proof of Claim 1: Let T be the transducer, and (v,m) be a position (in $\widetilde{G}$) reachable under the sequence α from which P_2 loses. We construct an extension α_2 of α such that all transducers T' that agree with α_2 behave identical to T. Since (v,m) is losing for P_2 in the game $\widetilde{G}$, this means that $\alpha_2\beta \notin \mathscr{L}_k(G)$ for every possible choice of β.

Let $\mathscr{T}'$ be the set of k-transducers that are compatible with α. Note that this set is finite as k is constant. Either all transducers $T' \in \mathscr{T}'$ behave identically to T after reading α (in which case we are done), or there is at least one transducer $T' = (M', \Sigma, \Gamma, s', \eta', L')$ that does not. If it does not, then there is a finite word $b = b_0 \ldots, b_n \in \Gamma^*$ of length at most k^2 such that feeding b to both T and T' from the respective states reached after α forces a different a output symbol of the two transducers T and T'. We now extend α to $\alpha' = \alpha\widehat{L}(\alpha|_\Gamma)b_0\widehat{L}(\alpha|_\Gamma b_0)b_1 \ldots \widehat{L}(\alpha|_\Gamma b_0 \ldots b_{n-1})b_n$ (recall that $\widehat{L}$ is the labeling function of the transducer extended to words).

The position (v', m') reached (in $\widetilde{G}$) under α' is still losing for P_2. This is because in the game $\widetilde{G}$, P_1's actions are fixed. Thus, all the positions reachable from positions that are themselves losing for P_2 still remain losing. However, when considering α' instead of α, the set $\mathscr{T}'$ does not contain T' any more. Note that since we only extended α, no new elements can be added to $\mathscr{T}'$ in this way. Since $\mathscr{T}'$ is finite, α' can be extended in this way until only transducers that behave identically to T are compatible with α'. The claim follows. □

Proof of Claim 2: If G is not k-transducer live, then there exists a prefix word α that is compatible with a k-transducer T such that no suffix word β exists so that $\alpha\beta$ induces a winning play for P_2 and $\alpha\beta$ can be generated by some transducer. In particular, for all suffixes $\beta \in A^\omega$ such that $\alpha\beta$ is compatible with T, the corresponding play is losing for P_2.

Let $\widetilde{G}$ be the game generated from T, and (v,m) be the position reached in $\widetilde{G}$ reached under α. Thus for all $\alpha\beta \in A^\omega$ compatible with T, the play β starting from (v,m) is losing for P_2. Moreover, any winning word ($\beta \in A^\omega$) for P_2 starting from (v,m) in $\widetilde{G}$ results in a word $\alpha\beta \in \mathscr{L}_k(G)$, which contradicts the assumption on α. This means that P_2 loses the parity game from (v,m). □

Adapting the above theorem to address the reachability case is simple, as the winning condition is only used for the fact that non-emptiness checking of a deterministic automaton is not harder than polynomial time. As a consequence of the above theorem, we can verify whether a given game is k-transducer live rather quickly. The following theorem shows that P_2 (the system) can always win games that are k-transducer live. Additionally, we also show that such a strategy (represented as a Turing machine) requires space only polynomial in k and the number of positions in the game. Overall, this shows that in k-transducer live games, the system can learn the behavior of the environment and adapt its own behavior without violating the objective encoded into the game.

Theorem 3. *Any k- transducer live game G is always winning for P_2. Furthermore, there exists a winning strategy for P_2 that requires at most polynomial space ($Poly(n,k)$), where n is the number of vertices in G. Moreover, if G is a reachability game, by following this strategy, P_2 can reach a final state in at most an exponential number of steps ($Poly(n)$, $EXP(k)$).*

Proof. We arrange the set of all possible transducers into a sequence $T_1, \ldots, T_N$ ($N \leq EXP(k)$) in lexicographic order.

Our strategy iterates over the transducers $T_i = (M_i, \Sigma, \Gamma, s_i, \eta_i, L_i)$ while generating a single sequence of decisions $b_0 b_1 \ldots \in \Gamma^\omega$. For every transducer T_i, the strategy checks if for the current vertex $v \in V$ in the game G played, the game $\widetilde{G}$ built according to the construction in the proof of Theorem 2 has position

(v,m) reachable for some $m \in M_i$. If that is not the case, this means that T_i cannot be the environment strategy, and the system player strategy moves to the next transducer T_{i+1} in the sequence. Otherwise, the strategy starts to maintain a set M' of transducer states that the transducer can currently be in. Initially, these are all states $m \in M_i$ for which (v,m) is reachable in $\widetilde{G}$.

Afterwards, the strategy picks one particular state $m \in M'$ as the current conjectured environment's transducer state. It computes a winning strategy from (v,m) in $\widetilde{G}$ in polynomial space and time, which is lasso-shaped. The strategy chooses the actions from this lasso while the actions of P_1 agree with the lasso, while also simultaneously updating the candidate set M' of current states. Once the actions of P_1 do not agree with the P_1 actions along the lasso any more, it is known that either the environment does not play T_i, or the current state of T_i is not m. State m is then removed from M'. If at some point M' becomes empty, the strategy moves to the next transducer T_{i+1}.

If a game is k-transducer live, the system always wins - eventually, the correct transducer T_i is found with the correct current state m, or P_2 manages to play a winning lasso already earlier. The winning strategy for P_2 against (T_i,m) ensures that P_2 wins when eventually, the (T_i,m) combination is considered. By k-transducer liveness, the prefix play until then does not prevent P_2 from winning once the correct (T_i,m) pair has been found. A corresponding strategy for reachability games works in the same way. For each (T_i,m), any deviating behavior of the environment is detected in at most $O(n \cdot k)$ many steps, as this is the maximum lasso length of the P_2 strategy computed in $\widetilde{G}$.

The number of transducer/state pairs is exponential in k. Hence, for reachability games, a final state is visited after a number of steps at most exponential in k and linear in n. □

In Theorem 2, we showed the upper complexity bound for detecting k-transducer liveness. The following theorem establishes co-NP-completeness for determining whether a Büchi game is k-transducer live.

Theorem 4. *Deciding whether a Büchi game G is k-transducer live is co-NP-hard.*

Proof. Let $\psi = C_1 \wedge C_2 \wedge \cdots \wedge C_r$ be a Boolean formula in conjunctive normal form (CNF) over the set $X = \{x_1, x_2, \ldots, x_k\}$ of variables. We construct a reachability game G_ψ corresponding to the formula ψ such that G_ψ is k-transducer live if and only if ψ is not satisfiable. The underlying game graph G_ψ is shown in Figure 3. The action set for P_1 is $\{\top_1, \bot_1, \top_2, \bot_2, \ldots, \top_k, \bot_k\}$. For $b \in \{\top, \bot\}$, P_1 uses the action b_i to assign the value b to the variable x_i. The action set for P_2 is $\{\varepsilon\}$, corresponding to a dummy move. Note that in this way, P_2 has no role to play.

We now prove that the formula ψ is satisfiable if and only if the game on G_ψ is k-transducer live. P_1 is restricted to k-transducer strategies. Since there are k variables, P_1 needs exactly one transducer state for each of the variables. In the game, P_1 makes an assignment for each variable, once per clause. P_2 repeats the same dummy (ε) move for each of its vertex. Say P_1 is currently assigning the variable x_i the value b for clause j, then the evaluation of this clause, denoted by $eval_j(i)$, is computed as $eval_j(i) = eval_j(i-1) \vee [\![C_j]\!]_{x_i=b}$. Here, $[\![C_j]\!]_{x_i=b}$ is $\top$ if setting x_i to b makes clause C_j satisfied, else it is $\bot$.

If clause j is satisfied, then the play moves on to clause C_{j+1} and P_1 starts assigning values to the set X once again. The fact that P_1 can only use k-transducer strategies implies that P_1 cannot change the assignments. If some clause is not satisfied, then the play moves to a green vertex and P_2 wins. On the other hand, if all the clauses are satisfied, then P_1 wins, i.e., the play reaches the orange vertex and can never reach a green vertex.

Thus, if ψ is satisfiable by a particular assignment for X, then P_1 uses this assignment to obtain the corresponding k-transducer to stay safe in G_ψ. Thus, G_ψ is not p-transducer live. Otherwise, if ψ is not satisfiable, there does not exist any k-transducer T for P_1 such that the play generated by the strategy f_T

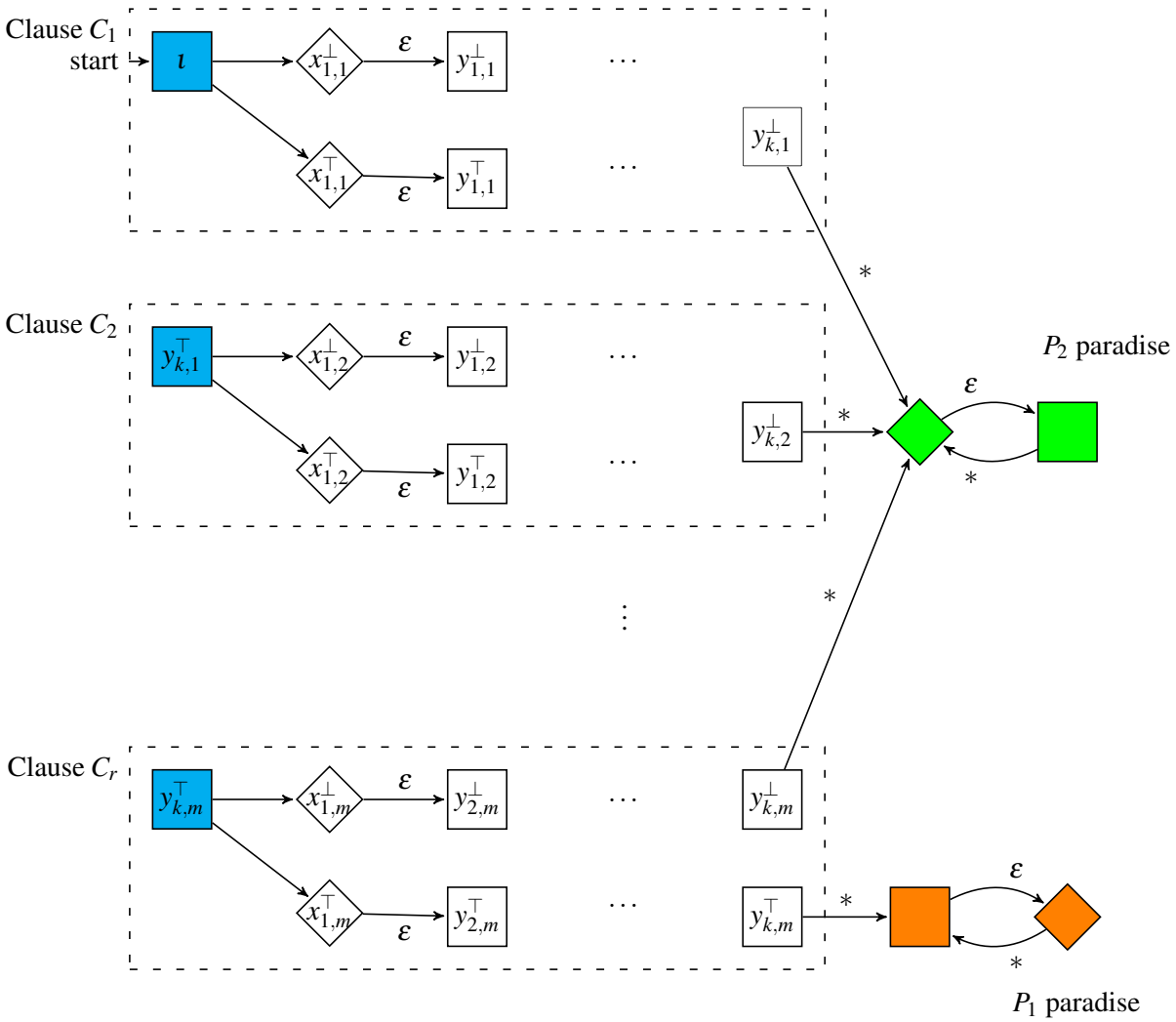

Figure 3: The game graph for the reachability game G_ψ corresponding to ψ. The initial vertex ι is marked as such. P_1 plays from squared vertices and P_2 plays from diamond-shaped vertices. The objective of P_1 is to reach the orange vertex in the P_1 paradise, i.e., satisfy the formula ψ. Dually, the objective of P_2 is to reach a green vertex in the P_2 paradise, i.e., falsify the formula ψ (however, there is no role for P_2 in the game, i.e., the moves of P_2 have no effect on the satisfaction of the clauses). Whether the current clause is already satisfied is represented in the superscript of the vertex label. From vertices with label $y^{\top}_{i,j}$ or $y^{\bot}_{i,j}$, P_1 makes the assignment for the next variable x_{i+1} (for clause j), using one of the actions $\top_{i+1}$ and $\bot_{i+1}$. Suppose P_1 makes the move $a \in \{\top_{i+1}, \bot_{i+1}\}$ from y^{α}_i for clause C_j, then the next state is $x^{\beta}_{i+1,j}$, where $\beta = \top$, if setting x_{i+1} to b makes clause C_j satisfied, and $\beta = \alpha$ otherwise.

does not eventually hit a green state. This is because the assignment corresponding to T cannot satisfy some clause C_j and the game hits the green vertex in the first clause where this happens. This implies that G_ψ is k-transducer live. □

The above proof of co-NP-hardness is similar to the proof of NP-completeness of decision problems for partial-observation games with mean-payoff objectives from [7] (Lemma 4). In [7], P_1 makes the assignment to the variables and P_2 chooses the clauses to check the assignment. We adopted this proof idea for the k-transducer liveness problem. The main difference is that in our reduction, we remove the role of P_2 and require P_1 to repeatedly make the same assignments for each clause. The fact that P_1 is restricted to k-transducers ensures that it makes the same assignments for each clause.

5 Conclusion

In this paper, we studied the problem of synthesizing controllers that satisfy given specifications when used in environments of a bounded size. While this problem was originally introduced by Kupferman Et al. [16], our work was motivated by applications in robotics. This problem has a simple definition and yet captures the idea that a high-level robot controller should operate correctly in environments with unknown dynamics of bounded complexity. The problem is also interesting on its own because it captures the idea that a controller may observe the environment's behavior to adapt to it.

We provided two results for this synthesis problem at the level of games, as reactive synthesis is commonly reduced to solving games. The game formulation enables us to understand a controller to be synthesized and the environment's behavior as strategies of the two players in the game, thereby simplifying the exposition. Our first result is negative: we strengthened the NP lower bound given by Kupferman et al. [16] to PSPACE. This PSPACE lower bound means that we cannot hope to employ a satisfiablity (SAT) solver for this problem. Such solvers have proven their applicability for a plethora of practical scenarios in the last two decades, so being able to use them would have helped to scale reactive synthesis under bounded environments to scenarios of practical interest.

We identified the system player's necessity to strategize to find out how it should probe the environment's behavior "safely" as the key reason for this high complexity. To address this issue, we defined the notion of k-transducer liveness, which captures games and reactive synthesis problem instances in which such strategic reasoning is not needed. Consequently, the system player only has to care about satisfying the objective encoded into the game once the environment transducer is safely found. We proved that this modified problem is co-NP-complete, thereby showing that its complexity is comparably lower. As an added benefit, our co-NP solving algorithm that is given in the proof of Theorem 2 can be implemented with a satisfiability (SAT) solver. After guessing an environment transducer, it performs an analysis of a graph that is the product of the guessed transducer and the given game. This graph a one-player game, which is conceptually the same as a deterministic automaton. Building such a product in a SAT instance is already done in exact SAT-based minimization of deterministic automata [2, 9], except that in our case, reachability of product game positions also needs to be considered. The positive results obtained for deterministic automaton minimization in the past suggest a reasonable scalability of SAT-based k-transducer liveness game solving and k-transducer liveness reactive synthesis.

Our work focused on identifying computational complexities to prepare high-level robotics applications in unknown environments. As such, we had to exclude some practical considerations from this work, which we leave for future work. Our approach for k-transducer live games computes strategies that run through all the possible environment transducers. This makes it slow to converge to the final behavior (for a fixed environment strategy). Finding ways of improving this approach will make solving

k-transducer live games more useful for practical applications. Although our choice to cast the complexity of the behavior of the environment transducer as its number of states is a natural one (in computer science), making our analysis of the problem interesting from a theoretical viewpoint, this choice can be further honed from a practical perspective. One way of doing this would be to synthesize a transducer that works correctly if the environment *eventually* follows a fixed transducer (where the system is not able to observe when this happens). This idea has already been applied for synthesizing implementations that are robust against deviations from environment assumptions [12, 10]. It was shown that due to the fact that the synthesis algorithms always compute finite-state implementations, these implementations have to start working towards the satisfaction of the specification even before the environment can be observed to have stabilized. A similar effect can be expected for synthesizing implementations that perform early best-effort specification satisfaction in environments of bounded complexity as well. Hence, analyzing the problem of reactive synthesis for environments that eventually follow a bounded transducer appears to be worthwhile.

References

[1] Bowen Alpern & Fred B. Schneider (1985): *Defining liveness*. *Inf. Process. Lett.* 21(4), pp. 181–185, doi:10.1016/0020-0190(85)90056-0.

[2] Souheib Baarir & Alexandre Duret-Lutz (2015): *SAT-Based Minimization of Deterministic ω -Automata*. In: *20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-20)*, pp. 79–87, doi:10.1007/978-3-662-48899-7_6.

[3] Roderick Bloem, Krishnendu Chatterjee & Barbara Jobstmann (2018): *Graph Games and Reactive Synthesis*. In: *Handbook of Model Checking*, Springer International Publishing, Cham, pp. 921–962, doi:10.1007/978-3-319-10575-8_27.

[4] Roderick Bloem, Rüdiger Ehlers & Robert Könighofer (2015): *Cooperative Reactive Synthesis*. In Bernd Finkbeiner, Geguang Pu & Lijun Zhang, editors: *Automated Technology for Verification and Analysis (ATVA)*, *Lecture Notes in Computer Science* 9364, Springer, pp. 394–410, doi:10.1007/978-3-319-24953-7_29.

[5] Allan Borodin & Ran El-Yaniv (1998): *Online computation and competitive analysis*. Cambridge University Press, Cambridge, England, UK.

[6] Romain Brenguier, Jean-François Raskin & Ocan Sankur (2017): *Assume-admissible synthesis*. *Acta Informatica* 54(1), pp. 41–83, doi:10.1007/s00236-016-0273-2.

[7] Krishnendu Chatterjee, Alexander Kößler & Ulrich Schmid (2020): *Automated analysis of real-time scheduling using graph games*. *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pp. 163–172, doi:10.1145/2461328.2461356.

[8] Rayna Dimitrova, Bernd Finkbeiner & Hazem Torfah (2019): *Synthesizing Approximate Implementations for Unrealizable Specifications*. In Isil Dillig & Serdar Tasiran, editors: *Computer Aided Verification*, Springer International Publishing, Cham, pp. 241–258, doi:10.1007/978-3-030-25540-4_13.

[9] Rüdiger Ehlers (2010): *Minimising Deterministic Büchi Automata Precisely Using SAT Solving*. In: *13th International Conference on Theory and Applications of Satisfiability Testing - (SAT)*, pp. 326–332, doi:10.1007/978-3-642-14186-7_28.

[10] Rüdiger Ehlers (2011): *Generalized Rabin(1) Synthesis with Applications to Robust System Synthesis*. In: *NASA Formal Methods - Third International Symposium, (NFM)*, pp. 101–115, doi:10.1007/978-3-642-20398-5_9.

[11] Rüdiger Ehlers, Robert Könighofer & Roderick Bloem (2015): *Synthesizing cooperative reactive mission plans*. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS)*, pp. 3478–3485, doi:10.1109/IROS.2015.7353862.

[12] Rüdiger Ehlers & Ufuk Topcu (2014): *Resilience to intermittent assumption violations in reactive synthesis*. In: *17th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pp. 203–212, doi:10.1145/2562059.2562128.

[13] Paul Hunter, Guillermo A. Pérez & Jean-François Raskin (2017): *Reactive synthesis without regret*. *Acta Informatica* 54(1), pp. 3–39, doi:10.1007/s00236-016-0268-z.

[14] Valerie King, Orna Kupferman & Moshe Y. Vardi (2001): *On the Complexity of Parity Word Automata*. In: *Foundations of Software Science and Computation Structures, 4th International Conference (FOSSACS)*, pp. 276–286, doi:10.1007/3-540-45315-6_18.

[15] Hadas Kress-Gazit, Morteza Lahijanian & Vasumathi Raman (2018): *Synthesis for Robots: Guarantees and Feedback for Robot Behavior*. *Annu. Rev. Control Rob. Auton. Syst.* 1(1), pp. 211–236, doi:10.1146/annurev-control-060117-104838.

[16] Orna Kupferman, Yoad Lustig, Moshe Vardi & Mihalis Yannakakis (2011): *Temporal Synthesis for Bounded Systems and Environments*. *Symposium on Theoretical Aspects of Computer Science (STACS2011)* 9, doi:10.4230/LIPIcs.STACS.2011.615.

[17] Rupak Majumdar, Nir Piterman & Anne-Kathrin Schmuck (2019): *Environmentally-Friendly GR(1) Synthesis*. In: *25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 229–246, doi:10.1007/978-3-030-17465-1_13.

[18] Daniel Neider (2011): *Small Strategies for Safety Games*. In: *Automated Technology for Verification and Analysis*, Springer, Berlin, Germany, pp. 306–320, doi:10.1007/978-3-642-24372-1_22.

[19] Christos H. Papadimitriou & Mihalis Yannakakis (2020): *On complexity as bounded rationality*. *Proceedings of the twenty-sixth annual ACM symposium on Theory of Computing*, pp. 726–733, doi:10.1145/195058.195445.

[20] Marco Pistore & Moshe Y. Vardi (2007): *The Planning Spectrum - One, Two, Three, Infinity*. *J. Artif. Intell. Res.* 30, pp. 101–132, doi:10.1613/jair.1909.

[21] Sven Schewe & Bernd Finkbeiner (2007): *Bounded Synthesis*. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino & Yoshio Okamura, editors: *Automated Technology for Verification and Analysis*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 474–488, doi:10.1007/s10009-012-0228-z.