Integration of Deterministic Inference with Formal Synthesis for Control under Uncertainty

Kevin J. Leahy[1], Prasanna Kannappan[2], Adam Jardine[3], Herbert Tanner[2], Jeffrey Heinz[3], and Calin Belta[1]

***Abstract*— In this work, we consider an agent playing a turn-based game in a known environment against an adversary with unknown dynamics. The model of the adversary is assumed to belong to a subclass of regular languages that can be learned in the limit. We use tools from formal methods to synthesize a control strategy for the agent to win the game as it learns the model of its adversary, if a winning strategy exists. The strategy is updated as new information about the adversary is learned. The proposed framework is tested in simulation.**

I. Introduction

Uncertainty is one of the main challenges in symbolic control of dynamical systems that interact with their environments. Uncertainty may be present in the dynamics of the system under control, its observations about its environment, the mechanisms of interaction with it, as well as the description of the environment itself. In this paper we focus on the last source of uncertainty, namely the description of the environment, and we specifically assume that this environment has its own autonomous dynamics which are unknown. In fact, we consider a case where the environment may behave in a way that is antagonistic to the system, actively trying to prevent the system from achieving its control specification.

Adversarial interactions between systems is often considered in the context of game theory. Algorithmic game theory [1], in particular, deals with theoretical and computational properties of games evolving on graphs, and provides several methods for determining winning strategies in such games. One–player stochastic games are equivalent to Markov Decision Processes (MDPs) [2]. In such a framework, uncertainty is typically taken into account by developing probabilistic models of player behavior, and reasoning in a Bayesian sense. There are distinctions, however, between *almost-surely* winning a game, which means that the player wins with some degree of certainty, and *sure-winning* a game, which means that the player is guaranteed to win, no matter what the adversary does [2]. Chatterjee and Henzinger [2] have shown that sure-winning strategies for two–player deterministic games are almost surely winning in one–player stochastic games but *not* vice versa. This paper departs from the Bayesian paradigm by treating uncertain dynamics in a purely non-probabilistic setting. One reason to prefer a non-probabilistic approach to reasoning and strategizing in the face of adversarial interaction with unknown dynamics is that an alternative Bayesian approach would require probabilistic priors, which may or may not be available. In addition, as [2] suggests, a non-probabilistic approach offers solutions that are also applicable to stochastic systems.

Uncertainty can in principle be resolved, at least partially, through learning or inference methods. Incorporating learning in control has been pioneered by extensive studies in Reinforcement Learning (RL). RL offers methods for *learning a control strategy* by formulating the control problem in an *uncertain* environment as an MDP [3], [4]. The difference between our approach and reinforcement learning is that in our work learning is decoupled from control design: the learner in our approach performs system identification, while a control design methodology of choice can then take over based on the identified model. Moreover, the same identified model can be used with any compatible control synthesis, yielding possibly different policies depending on what purpose is to be served each time. In other words, the (intermediate) outcome of the learner here can actually be recycled and reused.

An important aspect of an algorithmic game theoretic approach to control synthesis in the case of adversarial interactions is that it places the problem inside the domain of formal languages, and thus allows the utilization of a whole new suite of analytical tools. It is this particular link that enables us here to borrow machine learning techniques that are rooted in language identification. The idea is that by observing traces, or strings, that are acceptable by some finite state machine, one is able to incrementally construct an increasingly more accurate model of that machine. While learning a formal language from only positive samples may be intractable in general, it has been shown to be possible to learn certain subclasses of regular languages in the limit [5], [6]. In this work we illustrate the approach with such a subclass, namely the Strictly k-Piecewise languages [7], to model the motion of the adversary. Thus, the inferred structure of the adversary's possible behavior converges asymptotically to the true structure.

Specifically, we consider an agent competing against an adversary with unknown dynamics but some a priori knowledge of the appropriate model space with which the

[1]Department of Mechanical Engineering, Boston University, Boston, MA 02215 USA (e-mail: {kjleahy,cbelta}@bu.edu).

[2]Department of Mechanical Engineering, University of Delaware, Newark, DE 19716 USA (e-mail: {prasanna,btanner}@udel.edu).

[3]Department of Linguistics and Cognitive Science, University of Delaware, Newark, DE 19716 USA (e-mail: {ajardine,heinz}@udel.edu).

This work was partially supported by NSF CNS-1035588 at Boston University.

adversarial dynamics can be modeled.[1] The agent is tasked with satisfying a mission specified as a linear temporal logic (LTL) formula while avoiding being captured by the adversary. To correctly satisfy the mission specification, the agent must learn the unknown dynamics of its adversary. Our method allows the agent to infer the model of its adversary's motion and incrementally update its strategy according to new information. Thus, although we still pose the problem in a game-theoretic framework, we (a) resolve uncertainty in a non-probabilistic manner, and (b) perform control synthesis using mainstream model checking methods.

The closest existing work that is related to this paper is that of Chen et al. [11] and Fu et al. [12]. Fu et al. [12] develop a game-theoretic approach to finding winning strategies when interacting with an unknown adversary, for cases of reachability [13] as well as for temporal logic [12] specifications. Chen et al. [11], on the other hand, consider an agent that has to learn the dynamics of a stochastic environment in order to satisfy an LTL mission specification. The authors of that work model the motion of the adversary as a Markov chain, using stochastic languages and probabilistic priors. Given some information about the structure of the adversary's model, Chen et al. [11] incrementally identifies the transition probabilities.

The main difference of this paper with respect to the approach of Chen et al. [11] is that here the adversary is modeled as an unknown *non-probabilistic* transition system. In some sense, this approach meets the challenge of interacting with an unknown adversary at an earlier step compared to the formulation of Chen et al. [11], by identifying the structure itself using observations of the adversary's behavior. The probability of the adversary taking any specific action is ignored. This is intuitively the reason why non-probabilistic policies, when available, work both with and without stochasticity. They are designed to perform even for the worst case, and thus, rather than ensuring winning with probability one, they can actually offer absolute performance guarantees. The difference of the present work compared to that of Fu et al. [12] is that here the synthesis of policies is done using standard model checking tools.

The organization of the rest of the paper is as follows. In §II, we introduce the models used in this work and formally state the problem of interest. In §III, we present the solution for synthesizing a control policy and updating it according to new information about the environment. Simulation results are presented in §IV, and finally, in §V we present our conclusions.

II. Models and Problem Formulation

Notation For two sets A and B, we denote their Cartesian product as $A \times B$ and set difference operation as $A \backslash B$. The cardinality and power set of A are denoted by $|A|$ and 2^A respectively. The set of finite sequences of symbols from A is written as A^*. Similarly, we write A^k and $A^{\leq k}$ to denote sequences of symbols from A with length equal to k and length up to k.

A. Agent Dynamics

Consider an agent ag_i operating on a graph environment $E = (V, \mathcal{E})$, where V is a set of vertices, $\mathcal{E}$ is a set of edges, $i = 1, \ldots, N$ corresponds to the index of an agent and N is the number of agents. We assume the existence of a labeling function $\mathscr{L} : V \to 2^{AP}$ that labels nodes of the environment from a finite set of atomic propositions AP. We model the motion of the agent in the environment E as a *deterministic transition system*.

Definition 1: A deterministic transition system is a tuple $T = \big(Q,\, \bar{Q},\, \Sigma \,\to,\, \Pi,\, \models\big)$ with

1) Q: a finite set of states;
2) $\bar{Q} \subseteq Q$: an initial state;
3) Σ: a finite set of actions
4) $\to$: $Q \times \Sigma \to Q$: a transition function;
5) Π: a finite set of atomic propositions;
6) $\models \subseteq Q \times \Pi$: a satisfaction relation.

We denote the transition system for agent ag_i as $T_i = \big(Q_i,\, \bar{Q}_i,\, \Sigma_i,\, \to_i,\, \Pi_i,\, \models_i\big)$, where $Q_i \subseteq V$. Further, the set of atomic propositions $\Pi_i \subseteq AP$ is a subset of the labels of the environment graph. For an atomic proposition $\pi_i \in \Pi_i$ and a state $q_i \in Q_i$, $(q_i, \pi_i) \in \models_i$ if and only if $\pi_i \in \mathscr{L}(q_i)$. That is, an agent satisfies a proposition by visiting a node in the environment that is labeled with that proposition.

Similarly, we model the motion of the adversary as a transition system $T_0 = \big(Q_0,\, \bar{Q}_0,\, \Sigma_0,\, \to_0,\, \Pi_0,\, \models_0\big)$, where the definitions of Q_0, $\bar{Q}_0$, $\to_0$, Π_0, and $\models_0$ are the same as for T_i. Unlike the agents, there is only one adversary present in the game.

It is important to note that for an agent or the adversary,

$$(q, \sigma, q') \in \to \;\Rightarrow\; (q, q') \in \mathcal{E}\ ,$$

which is to say that in general, when either an agent or the adversary chooses an action σ, the resulting move may be across several edges in the graph environment. Thus, even though T_i and T_0 are deterministic, inputs Σ_i and Σ_0 are required to indicate that both an agent and the adversary may move across multiple regions in the environment.

B. Agent Interaction

The agents and the adversary play a turn-based game in the environment E. The order in which they play their moves is predetermined and does not change during the game. For simplicity sake, lets assume that the adversary plays first followed by agent ag_1, agent ag_2, ..., agent ag_N (in the order specified), then the turn returns to the adversary and the cycle continues. An agent or the adversary chooses a transition from $q \in Q_i$ to some $q' \in Q_i$ such that $(q, q') \in \to_i$, followed by the other player making its own choice of transition. The mechanics of this behavior of playing in turns are captured by the *turn-based product*, which is adapted here to apply to transition systems.

Definition 2 (Turn-based product): For two players $T_0 = (Q_0, \bar{Q}_0, \Sigma_0, \to_0, \Pi_0, \models_0)$ and $T_1 = (Q_1, \bar{Q}_1, \Sigma_1, \to_1$

[1] If there is zero a priori knowledge then "no free lunch" theorems show that learning is not feasible [8], [9], [10].

$, \Pi_1, \models_1)$, their turn-based product is another transition system $T = (Q, \bar{Q}, \Sigma, \rightarrow, \Pi, \models)$ defined as follows:

1) $Q = Q_0 \times Q_1 \times \{\mathbf{0}, \mathbf{1}\}$, where $\mathbf{t} \in \{\mathbf{0}, \mathbf{1}\}$ marks whose turn it is: $\mathbf{0}$ for player 0 and $\mathbf{1}$ for player 1.
2) $\bar{Q} = \bar{Q}_0 \times \bar{Q}_1$.
3) $\Sigma = \Sigma_0 \cup \Sigma_1$.
4) $\rightarrow \subseteq \ Q \times \Sigma \times Q$ is the transition relation, according to which $((q_0, q_1, \mathbf{0}), \sigma_0, (q_0', q_1, \mathbf{1})) \in \rightarrow$ if $(q_0, \sigma_0, q_1') \in \rightarrow_1$, and $((q_0, q_1, \mathbf{1}), \sigma_1, (q_0, q_1', \mathbf{0})) \in \rightarrow$ if $(q_1, \sigma_1, q_1') \in \rightarrow_1$.
5) $\Pi = \Pi_0 \cup \Pi_1$.
6) $(q_0, q_1, \mathrm{t}) \models \pi \in \Pi$, for $(q_0, q_1, \mathrm{t}) \in Q_0 \times Q_1 \times \{\mathbf{1}, \mathbf{0}\}$, if either $\pi \in \Pi_0$ and $q_0 \models_0 \pi$, or $\pi \in \Pi_1$ and $q_1 \models_1 \pi$.

This definition can be easily extended to any number of players. In this problem formulation, we have $N+1$ players (N agents and 1 adversary). Implicit in this definition is the assumption that one player does not directly interfere with the behavior of the other, (e.g., blocking some of its transitions). Resolution of this type of interference, when present, takes place at the level of synthesis.

We assume that all agents have complete knowledge of their own transition system, the action set of adversary and the vertices in the environment where the other agents and adversary might be located but no knowledge adversary's transition relation. That is, an agent has knowledge of Q_0 and Σ_0 but no knowledge of $\rightarrow_0$. The agents also have some apriori knowledge about the class of formal languages to which the adversary's behavior belongs. We also assume that all agent can observe the location of the adversary at all times, regardless of their relative position with respect to any player in the environment. With these assumptions, the agents are naive about the possible behavior of the adversary, but observe all behavior of the adversary when it moves in the environment.

C. Temporal Logic Mission Specification

In this work we consider missions to be carried out by the agent that can be specified as LTL formulas. Given a set of atomic propositions AP, such formulas are defined recursively as

$$\varphi = p | \neg\varphi | \varphi_1 \vee \varphi_2 | \varphi_1 \wedge \varphi_2 | \Diamond\varphi | \Box\varphi | \varphi_1 \mathcal{U} \varphi_2 | \bigcirc \varphi \,,$$

where $p \in AP$ is an atomic proposition, and φ, φ_1, and φ_2 are LTL formulas. LTL combines the Boolean operators $\neg$ (negation), $\vee$ (disjunction), and $\wedge$ (conjunction) with the temporal operators $\Diamond$ (eventually), $\Box$ (always), $\mathcal{U}$ (until), and $\bigcirc$ (next). The semantics of LTL are given over infinite words from the set 2^{AP}. By combining operators from LTL, complex specifications can be created. A complete description of the syntax and semantics of LTL is given in Baier and Katoen [14].

For a turn-based product, we consider missions specified as LTL formulas over the set of atomic propositions Π, as defined in §II-B. These formulas can be used to specify persistent tasks φ_{pers} over Π_1 for the agent such as

$$\varphi_{pers} = \Box\Diamond\pi_{11} \wedge \Box\Diamond\pi_{12} \wedge \Box\neg\pi_{1obs} \,,$$

which can be expressed in English as "visit regions π_{11} and π_{12} infinitely often and never visit region π_{1obs}."

Similarly, winning conditions of the game for the agent can be given as an LTL specification φ_{game} over Π. For simplicity of presentation, we consider a set $\Pi_G \subseteq 2^{\Pi}$, where atomic propositions $\pi_G \in \Pi_G$ label winning or losing states of the turn-based product[2]. If for example we define $\pi_{capture} \in \Pi_G$ as the proposition used to label states in which the agent and the adversary occupy the same region of the environment—that is, $\pi_{capture}$ is the set of all $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$ such that $\pi_1 \wedge \pi_2$ implies the agent and adversary occupy the same region—then an example of a game formula is

$$\varphi_{game} = \Box\neg\pi_{capture} \,,$$

which is expressed in English as "never occupy the same state as the adversary." Other winning conditions can also be expressed in this manner, such as "occupy the same state as the adversary infinitely often." Then, with φ_{pers} and φ_{game} thus defined, we can define the class of all formulas we consider as

$$\varphi = \varphi_{pers} \wedge \varphi_{game} \,.$$

These formulas then capture both the persistent tasks for the agent with respect to the regions of the environment and the restrictions on its movement with respect to the adversary.

D. Problem Formulation and Solution Outline

We have now presented the necessary preliminaries to formally state the problem under consideration:

Problem 1: Given an agent modeled as transition system T_1 and an adversary modeled as transition system T_0 with unknown dynamics whose interactions are modeled with a turn-based product T, and a mission specification φ, synthesize a control policy such that the agent satisfies φ in the limit if such a policy exists.

The solution can be summarized as follows. First, at each time step, we infer a grammar based on any newly observed behavior of the adversary, as presented in §III-A. Given an inferred model for the behavior of the adversary, we construct a control policy for the agent to satisfy the specification. The synthesis of such a control policy is presented in §III-B. When the grammar describing the behavior of the adversary is updated, we update the control policy for the agent (§III-C).

III. Problem Solution

A. Learning Agent Interaction

We will now present the method for learning the structure of T_0. There are many types of a priori knowledge that can be assumed about the behavior of the adversary that lead to successful learning [6]. To be concrete, we assume that the adversary exhibits a behavior that can be expressed by a k-Piecewise formal language. Informally, a language $L \subseteq \Sigma^*$

[2]In general, it is possible to specify such conditions as formulas over Π, but such specifications are more difficult to interpret. Hence, our definition of Π_G to aid the reader.

is Strictly k-Piecewise if each word w's membership in L can be determined by checking each *subsequence* of length k of w. A subsequence of w of length k is any sequence of k symbols which appears in w (not necessarily contiguously). For details on the formal definition of such languages, see the Appendix.

We will assume here that $k = 2$. We thus can discover how each single action of T_0 may constrain its future actions. To put things in perspective, if $k > 2$, we can discover how each (non-contiguous) sequence of $k - 1$ actions may constrain T_0's next move. Increasing k allows for encoding complex constraints and specifications, which is in principle possible but will not be attempted here for simplicity of exposition.

Following Gold [5], we imagine a learner as a function ϕ that maps experience to grammars, i.e., $\phi : \mathcal{T}_{\text{fin}} \to \mathbb{G}$, where $\mathcal{T}_{\text{fin}}$ is a finite sequence of words from Σ_0^* and $\mathbb{G}$ is a class of grammars. An infinite sequence of words from Σ_0^* is a text $\mathcal{T}$ for language $L \subseteq \Sigma_0^*$ if every word of L occurs at least once in $\mathcal{T}$. For each $i \in \mathbb{N}$, let $\mathcal{T}[i]$ denote the finite initial portion of text $\mathcal{T}$ up to and including position i. We say that a learner converges on a text $\mathcal{T}$ if there exists a *convergence point* $p \in \mathbb{N}$ and a grammar G such that for all $i > p$, $\phi\big(\mathcal{T}[i]\big) = G$. A learner ϕ identifies a language L in the limit from text if for any text $\mathcal{T}$ for L, ϕ converges on $\mathcal{T}$ to grammar G and $L(G) = L$. If a learner can identify in the limit any language L belonging to a class of languages $\mathcal{L}$, then we say that ϕ identifies the class $\mathcal{L}$ in the limit.

In the case of Strictly k-Piecewise languages, one learner takes a very simple and intuitive form [6]:

$$\phi_k\big(\mathcal{T}[i]\big) = \begin{cases} \varnothing & i < 0 \\ \phi_k\big(\mathcal{T}[i-1]\big) \cup f_k\big(\mathcal{T}[i]\big) & \text{otherwise}\,, \end{cases}$$

where $\varnothing$ denotes the empty set and f_k returns the k-long subsequences of w (see Appendix). In other words, grammars can be thought of as well-formed subsequences of length k and learning proceeds by collecting observations of these k-long subsequences.

Strictly k-Piecewise languages are identifiable in the limit from positive presentation in time $\mathcal{O}(n^k)$, where n is the sum of lengths of all the strings in the presentation [15].

Example 1: To illustrate how a learner operates, we present the following example. First, we assume the adversary moves on a grid environment and Σ_0 are the inputs which would allow it to transition from one cell in this environment to an adjacent one along any one of the four compass directions. No diagonal cell transitions allowed, but it may be possible to transition over several cells in one move along a given compass direction. How many cells it can move along a given direction is not known a priori. In fact, with the exception of the impossibility of diagonal motion, and of the inclusion of the language of T_0 into the class of Strictly 2-Piecewise languages, no other prior knowledge about the adversary is assumed. The learner's initial hypothesis about the language of the adversary is that $L = \varnothing$, essentially implying that it does not move. As the adversary starts moving, the learner's hypothesis will be updated and refined based on the observations of transitions between adjacent cells. These transitions are associated with elements in Σ_0.

We will denote the set of compass directions $C = \{\mathsf{N}, \mathsf{S}, \mathsf{E}, \mathsf{W}\}$, and let $\Sigma_0 = C \times \mathbb{N}$, with the second element expressing the number of cells that the agent moved in the particular compass direction. Let the class of grammars $\mathbb{G}$ be the finite powerset of $\Sigma_0^{\leq 2}$. For any $G \in \mathbb{G}$, let $L(G)$ be the set of all and only those words w such that the 2-long subsequences of w are all contained in G (see Appendix). The language class $\{L(G) \mid G \in \mathbb{G}\}$ is a class that can be identified in the limit from positive presentation [7].

As the learner ϕ observes a sequence $w \in \Sigma_0^*$, it outputs grammars from text $\mathcal{T}$ as follows.

$$\phi\big(\mathcal{T}\big) = \{v \mid (\exists w \in \mathcal{T})[v \text{ is a 2-long subsequence of } w]\}$$

For example, say that $(\mathsf{E}, 4)(\mathsf{E}, 4)$ is not present in the grammar of the adversary. This means that once it moves four cells to the east once, it cannot do so again, as this 2-subsequence is not allowed by its grammar. The learner will never observe the subsequence $(\mathsf{E}, 4)(\mathsf{E}, 4)$ in the sequence of actions by the adversary, and thus its grammar will never be updated to contain this subsequence.

The table below shows how the presence of a symbol in Σ_0 in an element of the grammar is interpreted in terms of the transition system of the agent. In other words, if you know $\sigma \in w$ for some $w \in G$ then you know something about a number of transitions in the transition system, as explained in the table below. If you observe the transition in the left column, this implies the logical proposition in the right column.

(N, n)	$\big[\forall (x,y), (x, y+n) \in Q_2; \quad \big((x,y),(x,y+n)\big) \in \to_2\big]$
(S, n)	$\big[\forall (x,y), (x, y-n) \in Q_2; \quad \big((x,y),(x,y-n)\big) \in \to_2\big]$
(E, n)	$\big[\forall (x,y), (x+n, y) \in Q_2; \quad \big((x,y),(x+n,y)\big) \in \to_2\big]$
(W, n)	$\big[\forall (x,y), (x-n, y) \in Q_2; \quad \big((x,y),(x-n,y)\big) \in \to_2\big]$

The grammar G can be used to learn the adversary's transitions system T_0. The only unknown component in T_0 is the transition relation $\to_0$. Define $\to_s$, a transition relation that captures all transitions for the adversary in a grid world obtained by executing actions in Σ_0 from every grid square. [7] provides a construction showing how grammar G can be expressed as a transition relation $\to_g$. The transition relation of the adversary $\to_0$ can be obtained as a product of $\to_s$ and $\to_g$.

$$\to_0 = \to_s \otimes \to_g \tag{1}$$

This product operation $\otimes$ between transition relations is analogous to product operation between two automata (see [16] for details).

Although our example illustrates the case in which the adversary can move in the four compass directions, this method of learning applies equally well to other types of movement, such as diagonal moves, or even moves that a knight in chess can make.

B. Control Policy Synthesis

After observing the behavior of the adversary and inferring a model for its behavior, we must synthesize a control

policy for the agent. To find behaviors satisfying an LTL specification φ, we construct a *Büchi automaton* $\mathcal{B}$ that accepts only those words which satisfy φ.

Definition 3: A Büchi automaton $\mathcal{B}$ is a tuple ($S_\mathcal{B}$, $\bar{S}_\mathcal{B}$, Σ, $\rightarrow_\mathcal{B}$, $\mathcal{F}_\mathcal{B}$) where

1) $S_\mathcal{B}$ is a finite set of states;
2) $\bar{S}_\mathcal{B} \subseteq S_\mathcal{B}$ is a set of initial states;
3) Σ is the input alphabet;
4) $\rightarrow_\mathcal{B}$: $S_\mathcal{B} \times \Sigma \mapsto 2^{S_\mathcal{B}}$ is a transition relation;
5) $\mathcal{F}_\mathcal{B} \subseteq S_\mathcal{B}$ is a set of accepting states.

A Büchi automaton accepts an infinite word over Σ if there exists at least one corresponding run in $\mathcal{B}$ that intersects with $\mathcal{F}_\mathcal{B}$ infinitely many times. There exist off-the-shelf tools such as LTL2BA [17] which allow efficient construction of Büchi automata from LTL formulas.

Given a turn-based product $T = \left(Q, \bar{Q}, \rightarrow, \Pi, \models\right)$ and a Büchi automaton $\mathcal{B} = \left(S_\mathcal{B}, \bar{S}_\mathcal{B}, \Sigma, \rightarrow_\mathcal{B}, \mathcal{F}_\mathcal{B}\right)$, we capture how the behavior of the game may satisfy the specification φ by constructing a *product automaton* $\mathcal{A}$ as follows.

Definition 4: The product automaton $\mathcal{A}$ is a tuple ($S_\mathcal{A}$, $\bar{S}_\mathcal{A}$, $\rightarrow_\mathcal{A}$, $\mathcal{F}_\mathcal{A}$) where

1) $S_\mathcal{A} = Q \times S_\mathcal{B}$ is a finite set of states;
2) $\bar{S}_\mathcal{A} = \bar{Q} \times \bar{S}_\mathcal{B}$ is a set of intial states;
3) Π: an input alphabet
4) $\rightarrow_\mathcal{A} \subset S_\mathcal{A} \times \Pi \times S_\mathcal{A}$ is a transition relation such that $(q, s) \times \pi \times (q', s') \in \rightarrow_\mathcal{A}$ if and only if $(q, q') \in \rightarrow$ and $(q, \sigma) \in \models$ and $(s, \sigma, s') \in \rightarrow_\mathcal{B}$;
5) $\mathcal{F}_\mathcal{A} = Q \times \mathcal{F}_\mathcal{B}$ is a set of accepting states.

Using the product automaton, it is possible to find the winning strategy, if one exists. First, we must define a cost scheme to measure the agent's progress towards satisfaction of the LTL formula, and, by extension, winning the game.

Typically, the distance between a state s in the product automaton and a state $s' \in \mathcal{F}_\mathcal{A}$ is computed as the shortest path on the automaton using Dijkstra's algorithm or another similar algeorithm. Because we are considering a gam transition system, Dijkstra's algorithm is insufficient, since it does not account for the antagonistic behavior of the adversary. We use a backward induction algorithm (Algorithm 1) that incorporates elements of minimax [18], in which the agent chooses the neighboring node that results in the shortest path—as in the usual algorithm—while the adversary chooses the neighboring node that results in the longest path. Such an algorithm captures the competing behavior of the agent and the adversary in the worst case for the agent.

Progress will be captured by *distance to acceptance*, $V(s)$ which is an energy-like function defined on the states of the product automaton as used in [19] to enforce satisfaction of and LTL formula. For a set $X \subseteq S_\mathcal{A}$, we say that X is *self-reachable* if all states in X can reach a state in X. That is $\forall x \in X$, $d(x, X) \neq \infty$. We denote the largest self-reachable subset of $\mathcal{F}_\mathcal{A}$ as $\mathcal{F}_{\mathcal{A}*}$. Function $V(s) = d(s, \mathcal{F}_{\mathcal{A}*})$ is the distance to acceptance from state s in the product automaton.Thus, V captures the minimum number of transitions to reach a self-reachable accepting state. By considering only paths to self-reachable accepting states, we ensure that we do not consider accepting states from which repeated satisfaction is impossible.

Algorithm 1: Backward Induction Algorithm

Input : Product automaton $\mathcal{A}$, Set of final states $\mathcal{F}$
Output: Distance d to final set
for $s \in S_\mathcal{A}$ **do**
  $d(s, \mathcal{F}) \longleftarrow \infty$;
for $s \in \mathcal{F}$ **do**
  $d(s, \mathcal{F}) \longleftarrow 0$;
$Q \longleftarrow S_\mathcal{A}$;
while $Q \neq \emptyset$ **do**
  $q \longleftarrow \arg\min_{s \in Q} d(s, \mathcal{F})$;
  $Q \longleftarrow Q \backslash \{q\}$;
  if *Adversary move* **then**
    $d(q, \mathcal{F}) \longleftarrow \max_{\{q' | (q, \pi, q') \in \rightarrow_\mathcal{A}\}} d(q', \mathcal{F}) + 1$;
  else
    if $d(q, \mathcal{F}) > \min_{\{q' | (q, \pi, q') \in \rightarrow_\mathcal{A}\}} d(q', \mathcal{F}) + 1$ **then**
      $d(q, \mathcal{F}) \leftarrow \min_{\{q' | (q, \pi, q') \in \rightarrow_\mathcal{A}\}} d(q', \mathcal{F}) + 1$

Once we have computed the distance to acceptance, we can find a control policy $\mu : S_\mathcal{A} \rightarrow S_\mathcal{A}$ that will guarantee to lead to satisfaction of φ from any state in the product automaton, should one exist. This policy is constructed by simply choosing the transition from the current state s that leads to a state s' with the lowest value of $V(s')$. For efficiency during execution of the control policy, the agent utilizes a control policy $\mu_T : Q \rightarrow Q$. This policy is obtained from μ by considering only those $(q, s_\mathcal{B})$ such that $s_\mathcal{B}$ is the current Büchi state. When a transition is enabled to a state $s'_\mathcal{B}$ in the Büchi automaton, the agent is given the policy for the new Büchi state. The control policy synthesis is summarized in Algorithm 2.

C. Control Policy Update

The product automaton $\mathcal{A}$ constructed in §III-B is guaranteed to accept strings that satisfy the specification φ. However, until the learner ϕ converges to the grammar G that describes exactly the behavior of the adversary, strategies devised in this way may be subject to preemption by this player. This is because there may exist actions of this adversary that have not been observed yet, and hence the available model T_0 may not be complete. The adversary may thus be in position to utilize one of these actions to block the run prescribed by the strategy. What can therefore be said about the outcome of the control synthesis process that takes place while the learner continues to identify the dynamics of the adversary, is that *whenever feasible* the control strategies will always satisfy the specification. The learning module, on its side, can guarantee that the model it provides for synthesis is the most complete one based on the history of observations of the adversary's behavior.

Once the learner ϕ converges to the true model T_0 of

Algorithm 2: Control Policy Synthesis

Input : Turn-based product T, Büchi automaton $\mathcal{B}$
Output: Product automaton $\mathcal{A}$, control policy μ
Construct $\mathcal{A}$ from T and $\mathcal{B}$;
Compute $d(s, \mathcal{F}_{\mathcal{A}})$ for all s in $S_{\mathcal{A}}$;
$\mathcal{F}_{\mathcal{A}*} \longleftarrow \mathcal{F}_{\mathcal{A}}$;
for $s \in \mathcal{F}_{\mathcal{A}*}$ **do**
  if $\min_{\{s' | (s,\pi,s') \in \rightarrow_{\mathcal{A}}\}} d(s', \mathcal{F}_{\mathcal{A}}) = \infty$ **then**
    $\mathcal{F}_{\mathcal{A}*} \longleftarrow \mathcal{F}_{\mathcal{A}*}/s$;
Compute $d(s, \mathcal{F}_{\mathcal{A}*})$ for all s in $S_{\mathcal{A}}$;
for $s \in S_{\mathcal{A}}$ **do**
  if $V(s) < \infty$ **then**
    $\mu(s) = \arg\min_{\{s' | (s,\pi,s') \in \rightarrow_{\mathcal{A}}\}} V(s')$;
  else
    $\mu(s)$ is undefined;
for $s = (q, s_{\mathcal{B}}) \in S_{\mathcal{A}}$ **do**
  if $s_{\mathcal{B}} \in S_{\mathcal{B}}$ *is current Büchi state* **then**
    $\mu_T(q) = \{q' \mid \mu(s) = (q', s_{\mathcal{B}})\}$

the adversary, we know that all accepting runs in $\mathcal{A}$ are also feasible. At this time, the synthesis algorithm becomes complete. The control synthesis essentially functions as it would if the dynamics of the adversary were known a priori. Until this stage is reached, the control synthesis module operates based on the best available model for the adversary. Whenever this model is refined by the learner ϕ as a result of some new capability of the adversary being observed, then the control synthesis module must update the control strategy. This process can, in general, be computationally intense; fortunately, for the classes of systems considered in this paper it can be performed incrementally, and thus faster.

To incrementally update the product automaton, we use an algorithm first presented in Vasile and Belta [20]. For each new transition (q, q') that ϕ adds to $\rightarrow_0$, the algorithm considers all $q \in Q_0$ from which such a transition may be made. For each of those states, a set of states in the product containing that state is maintained. Further, a set of transitions in $\rightarrow_{\mathcal{A}}$ from those states to states containing q' is created. From these two sets, the product automaton can be efficiently updated. For complete details of the algorithm, the reader is directed to Vasile and Belta [20].

IV. Simulations and Results

To test our algorithm, a game simulation with two agents and adversary operating in a grid environment[3] was developed (Fig. 1). The agents (shown as red and blue circles) must carry out the specification

$$\varphi = \Box\Diamond\pi_1 \wedge \Box\Diamond\pi_2 \wedge \Box\neg\pi_{capture}\,, \tag{2}$$

which translates into English as "visit regions π_1 (shown in violet) and π_2 (shown in yellow) infinitely often and always avoid capture by the adversary (shown in green)" The agents' motion primitives

$$\Sigma_1 = \Sigma_2 = \{(N,1),(S,1),(E,1),(W,1),(NE,1),(NW,1),(SE,1),(SW,1),(O,0)\} \tag{3}$$

allow them to transition one grid square in north, south, east, west, north-east, north-west, south-east, and south-west directions or just continue to stay in place respectively. In this example, the adversary's has same motion primitives as the agents, $\Sigma_0 = \Sigma_1 = \Sigma_2$. The difference between the agent and the adversary here is that the sequence of moves played by the adversary during the game or in technical terms, the language of the adversary belongs to a class of Strictly 2-Piecewise languages where the adversary is forbidden to move along the four compass directions more than once. In other words, the adversary's actions cannot have 2-subsequences $(N,1)(N,1)$, $(S,1)(S,1)$, $(E,1)(E,1)$ or $(W,1)(W,1)$. In this case, the Strictly 2-Piecewise grammar G of the adversary's language $L(G)$ is

$$G = (\Sigma_0 \times \Sigma_0) \backslash G_f. \tag{4a}$$

$$G_f = \{(N,1)(N,1),(S,1)(S,1),(E,1)(E,1),(W,1)(W,1)\} \tag{4b}$$

However there are no restrictions on the stay in place or diagonal moves for the adversary. Initially, the agents only knows that the adversary's language belongs to a class of Strictly 2-Piecewise language but have no knowledge of the adversary's transition relation $\rightarrow_0$ or the forbidden subsequences in the grammar of the adversary's language. By observing the actions of the adversary, the agents incrementally build a model of the adversary and devise a strategy to satisfy their specification.

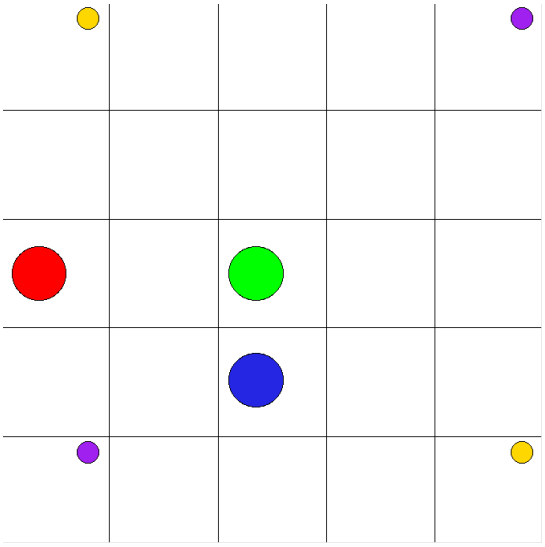

Fig. 1: Image of the 5×5 grid simulation game showing the agent (red circle), adversary (green circle) and the regions π_1 and π_2 shown labeled by violet and yellow circles, respectively.

[3]The grid structure is adopted here for illustration purposes only. The method is applicable to workspaces with arbitrary graph structures.

During the game, adversary and agents take turns to play. Each move comprises an adversary's turn followed by turn of agent ag_1 and then turn of agent ag_2. A move $m_i = (\sigma_i^{ad}, \sigma_i^{ag_1}, \sigma_i^{ag_2})$ consists of a tuple of an adversary action σ_{ad_i}, agent ag_1 action $\sigma_i^{ag_1}$ and agent ag_2 action $\sigma_i^{ag_2}$, where $\sigma_i^{ad} \in \Sigma_0$, $\sigma_{ag_1} \in \Sigma_1$, $\sigma_{ag_2} \in \Sigma_2$ and i refers to move number. We assume that at each move, the agents are able to observe the action of the adversary. After observing the adversary's action σ_i^{ad} in each move, the agents update a single centralized policy μ_T. During their respective turns, the agents then use the updated policy to determine a target state q' to transition to from the current state q, where $\mu_T(q) = q'$ and $q, q' \in Q$. If no winning policy exists from the current state, the agents continue to stay in place by executing the action $(O, 0)$. During each move, the adversary's action σ_i^{ad} is uniformly sampled from a subset of motion primitives $\Sigma_{L_i} \subseteq \Sigma_0$. If the adversary needs to choose an action for p$^{\text{th}}$ move (move number $m_i = p$) then Σ_{L_p} is given as,

$$\Sigma_{L_p} = \{x : x \in \Sigma_0, w = \sigma_1^{ad}\sigma_2^{ad}\dots\sigma_{p-1}^{ad}x, f_2(w) \in G\} \tag{5}$$

where f_k is a function that generates all possible subsequences of length k that can be found in the word w. (see Appendix)

Using the observed positive presentations of adversary's actions, the agent's Strictly 2-Piecewise learner ϕ_2 incrementally learns the grammar G corresponding to language of adversary's actions L in the limit (see §III-A for details). In other words, we can show that, if $\bar{\mu}_T$ is the policy computed when full knowledge about the adversary is available to the agent and μ_{Ti} is the policy computed at move i, then

$$\mu_{Ti_{i\to\infty}} = \bar{\mu}_T. \tag{6}$$

Define D_{pd}, a numerical distance measure that captures the difference between two policies.

$$D_{pd}(\mu_{Ti}, \bar{\mu}_T) = \Sigma_{q\in Q} d_q \tag{7a}$$

$$d_q = \begin{cases} 1, & \text{if } \mu_T(q) \neq \bar{\mu}_T(q) \\ 0, & \text{otherwise} \end{cases} \tag{7b}$$

Using (6) and (7), the convergence in grammar G corresponding to language of adversary's actions L in the limit can be written as,

$$D_{pd}(\mu_{Ti}, \bar{\mu}_T)_{i\to\infty} = 0 \tag{8}$$

To validate the result in (8), we run a series of 10 trials. Multiple runs of the game are required per trial as a single game can only produce a small subset of the 77 subsequences ($|G| = 77$, see (4a)). Furthermore, since the adversary's language belongs to the class of Strictly 2-Piecewise languages, each action of the adversary constrains its future actions. It was often observed that after a large number of moves (move number $m_i > 15$) in a game run, all future moves for the adversary led to the learner ϕ_2 acquiring minimal or no new knowledge about the adversary. Hence length of a game was fixed as 15 and 20 game runs were included in each trial. During each move of a game trial, indexed by m_i, D_{pd} is computed. Curves of color in Figure 2 show the variation in D_{pd} with number of moves seen by the learner (cumulatively added across multiple game runs) in a trial. The mean of D_{pd} over all trials, shown as a thick black line, demonstrates the convergence of D_{pd} to 0 as number of moves increases. The initial value of D_{pd} was 77 in this scenario, compared to over 45,000 states in Q.

From game simulations results (shown in Fig. 2), we can see that the policy μ_T incrementally learned by the agent through observations of adversary actions eventually converges to policy $\bar{\mu}_T$, the policy that would be computed if full knowledge of the adversary is available to the agent. Thus the grammar G learned by the Strictly 2-Piecewise learner ϕ_2 converges to the grammar of the adversary's action language L as the agent observes the behavior of the adversary for a sufficiently large number of moves. The control policy learned here would correspond to a strategy that would enable the agent to satisfy the game specification, provided such a strategy exists.

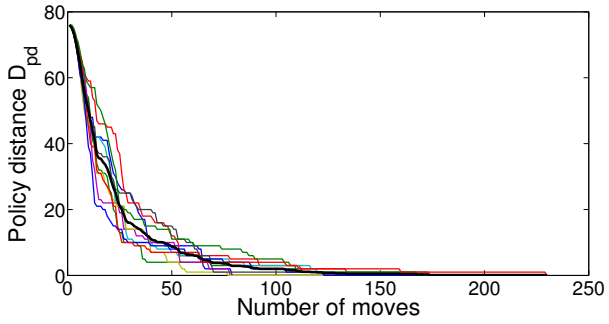

Fig. 2: The curves in color denote the evolution of D_{pd} w.r.t. the number of moves, for different game trial runs. The solid thick black curve represents the average of D_{pd} over all trials.

The particular implementation shown for this example is developed in Python. In this form, it can interface directly with physical hardware (e.g., quadrotors) in the same way as in Ulusoy and Belta [21].

V. Conclusion

Provably correct symbolic control synthesis can be feasible even in cases where the regulated system interacts with an unknown, but rule-governed adversary. In this setting, the system and the adversary are modeled as deterministic transition systems, and a model for the unknown adversary dynamics is constructed incrementally based on observations of its motion. Control synthesis can be performed using standard model checking tools, based on an evolving hypothesis about the dynamics of the adversary—the best hypothesis that can be formulated based on specific prior knowledge about the class of models the adversary model can belong to, and available data. The evolving model model can be shown to converge to the true dynamics in the limit, once a sufficient sample of adversary behavior has been observed, in which case control policies become as effective as those constructed

with full knowledge of the adversary dynamics. The paper demonstrates this idea with an small-scale example in which the adversary dynamics are captured by a formal language belonging in a subcategory of regular languages known as Strictly k-Piecewise languages. The latter class of languages is merely one of a very rich family of formal (subregular) language classes that can be treated within this framework of identification in the limit [6], [22]. Additional learning results along these lines are available in the context of grammatical inference [10].

APPENDIX

Strictly k-Piecewise Languages

The class of languages we consider forms a proper subset of regular languages and can be understood as follows. Different characterizations of this class exist [7], [15]. First let $\mathcal{P}_{\text{fin}}(A)$ denote the set of all finite subsets of some other set A. Let $|w|$ denote the length of a string w. Recall that $\Sigma^{\leq k}$ denotes the set of all strings of length no more than k made of symbols from Σ. A string $v = \sigma_1\sigma_2...\sigma_k$ is a *subsequence* of a string w, written $v \sqsubseteq w$, if there exists a sequence of strings $v_0, v_1, ..., v_k$ such that $w = v_0\sigma_1 v_1\sigma_2...\sigma_k v_k$.

For all $w \in \Sigma^*$, define $f_k(w) = \{v \in \Sigma^{\leq k} \mid v \sqsubseteq w\}$. The image of a string w under f_k is called the set of k-subsequences of w. The domain of f is lifted to sets of strings in the usual way: $f_k(L) = \bigcup_{w \in L} f_k(w)$.

Henceforth, we will introduce the necessary notation in the context of transition system T_0 with the alphabet Σ_0. A language L is Strictly k-Piecewise if for all $w \in L$, there exist a set $G \subseteq f_k(\Sigma_0^*)$ such that

$$w \in L \iff f_k(w) \subseteq G \ .$$

In other words, the set G is a grammar which contains the well-formed k-subsequences. Note G is necessarily finite. The words in L can be also given as follows.

$$L(G) = \{w \in \Sigma_0^* \mid f_k(w) \subseteq G\}$$

The notation $L(G)$ emphasizes that the above expression represents a mechanism for generating a (possibly infinite) language from a (finite) grammar.

REFERENCES

[1] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata logics, and infinite games: a guide to current research*. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[2] Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78:394–413, 2012.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[5] M. E. Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[6] J. Heinz. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[7] James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language*, volume 6149 of *Lecture Notes in Artifical Intelligence*, pages 255–265. Springer, 2010.

[8] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[9] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[10] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[11] Y. Chen, J. Tumova, A. Ulusoy, and C. Belta. Temporal logic robot control based on automata learning of environmental dynamics. *The International Journal of Robotics Research*, 32(5):547–565, 2013.

[12] J. Fu, H. G. Tanner, J. N. Heinz, K. Karydis, J. Chandlee, and C. Koirala. Symbolic planning and control using game theory and grammatical inference. *Engineering Applications of Artificial Intelligence*, 37:378–391, 2015.

[13] J. Chandlee, J. Fu, K. Karydis, C. Koirala, J. Heinz, and H. G. Tanner. Integrating grammatical inference into robotic planning. In *Proceedings of the 11th International Conference on Grammatical Inference*, number 21 in JMLR: Proceedings Track, pages 69–83, 2012.

[14] C. Baier and J.P. Katoen. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[15] Jeffrey Heinz. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906, Uppsala, Sweden, July 2010.

[16] Christos G. Cassandras. *Introduction to discrete event systems*. Springer Science and Business Media, 2008.

[17] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65, Paris, France, July 2001. Springer.

[18] T.H. Cormen. *Introduction to Algorithms*. MIT Press, 2009.

[19] X. Ding, M. Lazar, and C. Belta. Ltl receding horizon control for finite deterministic systems. *Automatica*, 50(2):399–408, 2014.

[20] C. I. Vasile and C. Belta. Sampling-based temporal logic path planning. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, pages 4817–4822, 2013.

[21] Alphan Ulusoy and Calin Belta. Receding horizon temporal logic control in dynamic environments. *The International Journal of Robotics Research*, 33(12):1593–1607, 2014.

[22] James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. Cognitive and sub-regular complexity. In Glyn Morrill and Mark-Jan Nederhof, editors, *Formal Grammar*, volume 8036 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.