2016 American Control Conference (ACC)
Boston Marriott Copley Place
July 6-8, 2016. Boston, MA, USA

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, and we specifically assume that the environment has its own autonomous dynamics which are unknown. In fact, we consider a case where the environment is antagonistic to the system.

Adversarial interactions are 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. One–player stochastic games are equivalent to Markov Decision Processes (MDPs) [2]. In such a framework, uncertainty is accounted for with probabilistic models of player behavior, and reasoning in a Bayesian sense. There are distinctions, however, between *almost-surely* winning a game and *sure-winning* a game, meaning the player wins 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. We depart from Bayesian methods by treating uncertain dynamics in a purely non-probabilistic setting. Such an approach is preferable to a Bayesian approach since it does not require probabilistic priors, which may or may not be available. In addition, as [2] suggests, non-probabilistic approaches offer solutions that are also applicable to stochastic systems.

[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.

Uncertainty may be resolved through learning or inference methods, such as 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]. In our work however, learning is decoupled from control design: the learner performs system identification, and a controller of choice can be used 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.

Algorithmic game theory places control synthesis inside the domain of formal languages, allowing the use of a whole new suite of analytical tools. This link enables us to borrow machine learning techniques rooted in language identification. By observing traces or strings that are acceptable by some finite state machine, we can incrementally construct an increasingly accurate model of that machine. While learning formal languages from only positive samples may be intractable in general, certain subclasses of regular languages can be learned 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 adversary's motion. 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 adversarial dynamics can be modeled.[1] The agent must satisfy a mission given as a linear temporal logic (LTL) formula while avoiding being captured by the adversary. To satisfy the specification, the agent must learn the adversary's unknown dynamics. 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 pose the problem in a game-theoretic framework, we resolve uncertainty in a non-probabilistic manner, and perform control synthesis using mainstream model checking methods.

The most closely related work 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] consider an agent that must learn the dynamics of a stochastic

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

environment to satisfy an LTL mission specification. The authors model the adversary's motion as a Markov chain, using stochastic languages and probabilistic priors. Given information about the structure of the adversary's model, they incrementally identify the transition probabilities.

The main difference between this work and [11] is that we model the adversary as an unknown *non-probabilistic* transition system. This approach meets the challenge of interacting with an unknown adversary at an earlier step compared to the formulation in [11], by identifying the structure itself via observations of the adversary's behavior, ignoring the adversary's probability of taking any specific action. This is intuitively the reason why non-probabilistic policies, when available, work both with and without stochasticity. They perform even for the worst case, and thus, rather than ensuring winning with probability one, they offer absolute performance guarantees. The difference of the present work compared to [12] is that our control synthesis is done using standard model checking tools.

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 E from a finite set of atomic propositions AP. We model the motion of the agent in E as a *deterministic transition system*.

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

1) $Q \subseteq V$: 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 TS for agent ag_i as $T_i = \left(Q_i, \bar{Q}_i, \Sigma_i, \to_i, \Pi_i, \models_i\right)$. Further, the set of atomic propositions $\Pi_i \subseteq AP$ is a subset of the labels of E. 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 E that is labeled with that proposition.

Similarly, we model the adversary's motion as a TS $T_0 = \left(Q_0, \bar{Q}_0, \Sigma_0, \to_0, \Pi_0, \models_0\right)$, 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 in the game.

Note that for an agent or the adversary,

$$(q, \sigma, q') \in \to \nRightarrow (q, q') \in \mathcal{E} .$$

That is, in general, when either an agent or the adversary chooses an action σ, the resulting move may be across several edges in E. 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 E. The order in which they play their moves is predetermined and does not change during the game. We assume that the adversary plays first followed by agent ag_1, agent ag_2, ..., agent ag_N, then the turn returns to the adversary and the cycle continues. Each player 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 we adapt 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 $T_1 = (Q_1, \bar{Q}_1, \Sigma_1, \to_1, \Pi_1, \models_1)$, their turn-based product is another TS $T = (Q, \bar{Q}, \Sigma, \to, \Pi, \models)$ 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) $\to \subseteq Q \times \Sigma \times Q$ is the transition relation, according to which $((q_0, q_1, \mathbf{0}), \sigma_0, (q_0', q_1, \mathbf{1})) \in \to$ if $(q_0, \sigma_0, q_1') \in \to_1$, and $((q_0, q_1, \mathbf{1}), \sigma_1, (q_0, q_1', \mathbf{0})) \in \to$ if $(q_1, \sigma_1, q_1') \in \to_1$.
5) $\Pi = \Pi_0 \cup \Pi_1$.
6) $(q_0, q_1, \mathbf{t}) \models \pi \in \Pi$, for $(q_0, q_1, \mathbf{t}) \in Q_0 \times Q_1 \times \{\mathbf{1}, \mathbf{0}\}$, 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 all agents have complete knowledge of their own TS, the adversary's action set, and the vertices in E where the other agents and adversary might be, but no knowledge of the adversary's transition relation. That is, an agent knows Q_0 and Σ_0 but not $\to_0$. The agents also have some a priori knowledge about the class of formal languages to which the adversary's behavior belongs. We also assume all agents observe the location of the adversary at all times, regardless of their relative position. With these assumptions, the agents are naive about the adversary's possible behavior, but observe its behavior when it moves in E.

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, which can specify complex missions. A complete description of the syntax and semantics of LTL is outside the scope of this work, but can be found in [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

$$\varphi_{pers} = \square\lozenge\pi_{11} \wedge \square\lozenge\pi_{12} \wedge \square\neg\pi_{1obs} \,,$$

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 2^{Π}. If we define $\pi_{capture} \in 2^{\Pi}$ as 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

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

which is expressed in English as "never occupy the same state as the adversary." Other winning conditions can be expressed in this manner, such as "occupy the same state as the adversary infinitely often." The class of all formulas we consider is:

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

These formulas 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 TS T_1 and an adversary modeled as TS 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. At each time step, we infer a grammar based on any newly observed adversary behavior. Given an inferred model for the adversary's behavior, we construct a control policy to satisfy the specification. When the grammar describing the adversary's behavior is updated, the control policy is also updated.

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 adversary's behavior that lead to successful learning [6]. We assume that the adversary's behavior can be expressed by a k-Piecewise formal language. Informally, a language $L \subseteq \Sigma^*$ 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 [7], [15].

We will assume that $k = 2$, and we can thus discover how each single action of T_0 may constrain its future actions. For $k > 2$, we could discover how each (non-contiguous) sequence of $k-1$ actions constrains T_0's next move. Increasing k allows for encoding more complex constraints, but will not be shown here for simplicity of exposition.

Following [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 [7], [15]). 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. These 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 adversary's language 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 power set 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 [7], [15]). 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.

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

For example, say that $(\mathsf{E}, 4)(\mathsf{E}, 4)$ is not present in the adversary's grammar. 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 TS 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 TS, as explained in the table below. If you observe the transition in the left column, this implies the logical proposition in the right column.

$$\begin{array}{ll}
(\mathsf{N}, n) & \left[\forall (x,y), (x, y+n) \in Q_2; \quad \big((x,y),(x,y+n)\big) \in \rightarrow_2\right] \\
(\mathsf{S}, n) & \left[\forall (x,y), (x, y-n) \in Q_2; \quad \big((x,y),(x,y-n)\big) \in \rightarrow_2\right] \\
(\mathsf{E}, n) & \left[\forall (x,y), (x+n, y) \in Q_2; \quad \big((x,y),(x+n,y)\big) \in \rightarrow_2\right] \\
(\mathsf{W}, n) & \left[\forall (x,y), (x-n, y) \in Q_2; \quad \big((x,y),(x-n,y)\big) \in \rightarrow_2\right]
\end{array}$$

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 $\rightarrow_0$. Define $\rightarrow_s$, 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 $\rightarrow_g$. The adversary's transition relation $\rightarrow_0$ can be obtained as a product of $\rightarrow_s$ and $\rightarrow_g$.

$$\rightarrow_0 = \rightarrow_s \otimes \rightarrow_g \tag{1}$$

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

This example illustrates the case in which the adversary can move in the four compass directions, but 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 adversary and inferring a model for its behavior, we synthesize a control policy for the agent. To find behaviors satisfying specification φ, we construct a *Büchi automaton* $\mathcal{B}$ accepting only words satisfying 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. Off-the-shelf tools such as LTL2BA [17] efficiently construct Büchi automata from LTL formulas.

Given a turn-based product $T = \big(Q, \bar{Q}, \rightarrow, \Pi, \models\big)$ and a Büchi automaton $\mathcal{B} = \big(S_\mathcal{B}, \bar{S}_\mathcal{B}, \Sigma, \rightarrow_\mathcal{B}, \mathcal{F}_\mathcal{B}\big)$, 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 initial 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 a similar algorithm. Because we consider a game TS, Dijkstra's algorithm is insufficient, as it does not account for the adversary's antagonistic behavior. We use a backward induction algorithm (Algorithm 1) incorporating 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, capturing the competing behavior of the agent and the adversary in the worst case for the agent.

Progress is captured by *distance to acceptance*, $V(s)$, an energy-like function defined on the states of the product automaton to enforce satisfaction of an LTL formula [19]. 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.

Once we compute the distance to acceptance, we can find a control policy $\mu : S_\mathcal{A} \rightarrow S_\mathcal{A}$ guaranteeing satisfaction of φ from any state in the product automaton, if one exists. 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

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**
      $d(q, \mathcal{F}) \leftarrow \min_{\{q' | (q,\pi,q') \in \rightarrow_{\mathcal{A}}\}} d(q', \mathcal{F}) + 1$

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.

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**
    $\mu_T(q) = \{q' \mid \mu(s) = (q', s_{\mathcal{B}})\}$

C. Control Policy Update

The product automaton $\mathcal{A}$ constructed in §III-B is guaranteed to accept strings that satisfy the specification φ. Until the learner ϕ converges to the grammar G that exactly describes the adversary's behavior, strategies devised in this way may be subject to preemption by this player. This is because there may be adversary actions not yet observed, 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. Therefore we can only say that while the learner continues to identify the adversary's dynamics, *whenever feasible* the control strategies will always satisfy the specification. The learning module 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 the adversary, all accepting runs in $\mathcal{A}$ are feasible. At this time, the synthesis algorithm is complete. The control synthesis essentially functions as it would if the adversary's dynamics 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. For complete details of such incremental updates, the reader is directed to [20].

IV. Simulations and Results

To test our algorithm, we simulate two agents and an adversary operating in a grid environment[2] (Fig. 1(a)). The agents (red and blue circles) must satisfy the specification

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

which translates into English as "visit regions π_1 (violet) and π_2 (yellow) infinitely often and always avoid capture by the adversary (green)" The agents' motion primitives allow them to transition one grid square in any direction, including diagonally, or to stay in place. The adversary has the same motion primitives as the agents ($\Sigma_0 = \Sigma_1 = \Sigma_2$). Unlike the agents, the adversary must play a sequence of moves belonging to a class of Strictly 2-Piecewise languages forbidding movement along the four compass directions more than once. That is, the adversary's actions cannot have 2-subsequences from $G_f = \{(N,1)(N,1), (S,1)(S,1), (E,1)(E,1), (W,1)(W,1)\}$.
In this case, the Strictly 2-Piecewise grammar G of the adversary's language $L(G)$ is

$$G = (\Sigma_0 \times \Sigma_0) \backslash G_f. \tag{3}$$

Initially, the agents only know the adversary's language belongs to a class of Strictly 2-Piecewise languages, with 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 adversary's actions, the agents incrementally build a model of the adversary and devise a strategy to satisfy their specification.

[2]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. A move $m_i = (\sigma_i^{ad}, \sigma_i^{ag_1}, \sigma_i^{ag_2})$ consists of a tuple of an adversary action $\sigma_{ad_i} \in \Sigma_0$, agent ag_1 action $\sigma_i^{ag_1} \in \Sigma_1$, and agent ag_2 action $\sigma_i^{ag_2} \in \Sigma_2$. At each move the agents observe the adversary's action and 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 stay in place by executing action $(O, 0)$. During each move, the adversary's action σ_i^{ad} is uniformly sampled from a subset of motion primitives permitted by its grammar.

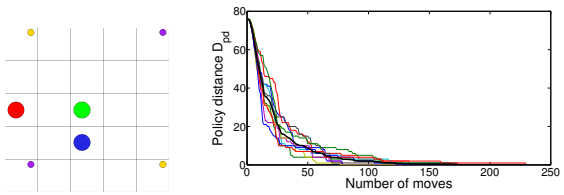
Using the observed adversary actions, the Strictly 2-Piecewise learner ϕ_2 incrementally learns the grammar G corresponding to the language of the adversary's actions L in the limit. That is, if $\bar{\mu}_T$ is the policy computed when full knowledge about the adversary is available to the agent and μ_{Ti} is the policy computed at move i, then

$$\mu_{Ti_{i\to\infty}} = \bar{\mu}_T. \tag{4}$$

We define $D_{pd}(\mu_{Ti}, \bar{\mu}_T)$ as the number of states in Q for which the policy μ_{Ti} differs from $\bar{\mu}_T$. The convergence in grammar G corresponding to language of adversary's actions L in the limit can be written as,

$$D_{pd}(\mu_{Ti}, \bar{\mu}_T)_{i\to\infty} = 0 \tag{5}$$

We ran a series of 10 trials. During each move of a game trial, indexed by m_i, D_{pd} is computed. The initial value of D_{pd} was 77 in this scenario, compared to over 45,000 states in Q. Curves of color in Fig. 1(b) show the variation in D_{pd} with number of moves seen by the learner. The mean of D_{pd} over all trials, shown as a thick black line, shows the convergence of D_{pd} to 0 as number of moves increases, implying that the policy incrementally learned by the agents eventually converges to policy $\bar{\mu}_T$. The learned control policy corresponds to a strategy enabling the agent to satisfy the game specification, if such a strategy exists.



(a) (b)

Fig. 1: Fig. (a) shows 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. In Fig. (b), 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.

V. Conclusion

Provably correct symbolic control synthesis can be feasible even in cases where a system interacts with an unknown, but rule-governed adversary. We model the system and adversary as deterministic transition systems, and incrementally construct a model for the unknown adversary dynamics by observing its motion. Standard model checking tools are used for control synthesis, based on an evolving hypothesis about the adversary's dynamics—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 model converges to the true dynamics in the limit, once enough adversary behavior has been observed, in which case control policies become as effective as those constructed with full knowledge of the adversary dynamics.

References

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

[2] K. Chatterjee and T. 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] R. S. Sutton and A. 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.

[7] J. Rogers, J. Heinz, G. Bailey, M. Edlefsen, M. Visscher, D. Wellcome, and S. Wibel. On languages piecewise testable in the strict sense. In *The Mathematics of Language*, volume 6149 of *Lecture Notes in Artifical Intelligence*, pages 255–265. Springer, 2010.

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

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

[10] C. 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*, volume 21, pages 69–83, 2012.

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

[15] 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.

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

[17] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *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.