Adaptive planning in unknown environments using grammatical inference

Jie Fu, Herbert G. Tanner and Jeffrey Heinz

***Abstract*— This paper presents a framework that integrates grammatical inference with symbolic control on finite-state transition systems interacting with partially unknown, adversarial, rule-governed environments. We model the interaction between a system and its environment as a two-player zero-sum game on graphs. We show that with some prior knowledge of the environment, the system can autonomously infer a game equivalent to the one actually being played and thus successively adapt its control strategy in polynomial time, and evolve its controller into one that ensures the completion of the desired task, whenever such a completion is possible.**

***Index Terms*— Grammatical inference, system identification, symbolic control, discrete event systems.**

I. INTRODUCTION

The paper demonstrates how grammatical inference (GI) [1] can support symbolic control design for finite-state transition systems operating in the presence of *unknown, and adversarial* environments. Such finite-state transition systems can arise as discrete abstractions of dynamical systems [2]–[5]. Synthesizing symbolic control plans becomes *adaptive* in the sense that the agent completes the information that is missing from its model about its world, and subsequently updates its plans, during execution time. The identification of the unknown environment model occurs asymptotically, based on observations of the interaction between agent and environment. Through this *learning* process, symbolic controllers are continuously refined until one that guarantees the satisfaction of task specifications is built after a finite number of steps.

In related literature, reactive synthesis has been introduced for control in the presence of dynamical environments, in which the system computes the control output based on real-time information [6]–[9]. In this work, an assumption on the environment is *given* [7], and the specification of the system is satisfiable provided the environment dynamics satisfy this assumption. Along these lines, an iterative planning framework is developed [10] for an agent in a partially unknown environment. When it is discovered that the environment imposes constraints, which render the current plan unrealizable, the agent replans. The discovered constraints are generated by the environment dynamics, however the dynamics itself is not identified. In this paper instead, it is the unknown environment dynamics that causes the system to fail. The only assumption made here about this dynamics is that it belongs to some general class.

We address the following question: is there a method to convert the problem of designing a symbolic controller for a system that interacts with an unknown adversarial environment, into a synthesis problem where environment dynamics is known?—then several known design solutions, e.g., algorithmic game theory [11] and discrete event system (DES) control theory [12], could be applied. A key observation is that once abstracted, the behavior of both agent and environment can be viewed as formal objects (automata, languages, grammars, etc.), and the identification of the environment model, and subsequently all possible interactions with it, is essentially a process of *inference*—to *generalize any* formal object that can describe this model based on the finite amount of observed behaviors. GI, as a sub-field of machine learning, is a paradigm that identifies formal objects through presentation of examples with or without a teacher [1], and thus the methodology naturally fits in this problem formulation.

Incorporating learning in control has been pioneered by studies on reinforcement learning (RL) and adaptive control. RL offers a method for *learning a control strategy* by formulating the control problem in an *uncertain* environment as a Markov decision process (MDP) [13]. The difference compared to the proposed GI-based framework is that in the latter, learning is decoupled from control design: GI performs system identification and the control design method of choice implements a strategy based on the identified model. This structure is reminiscent of the synergy between adaptation and control in continuous dynamical systems.

Some earlier work [14] employs automata learning to construct an MDP that captures the interaction between the system and its environment. Alternatively, a game formulation can be developed [15] for cases where the environment is *not* stochastic and unintentional, but deterministic and adversarial. There an assumption is made that the system cannot inhibit the dynamics of the environment. In addition, both aforementioned solutions need to produce an automaton [15] or a Markov chain [14], a fact that restricts the classes of learning algorithms that can be applied. In this paper we relax these assumptions: the environment behavior can be constrained by the actions of the system, and more importantly, the learner does not have to produce an automaton as a model for the unknown adversary. Instead, what is learned is the interaction itself: we learn the *game* being played between the system and its environment, up to some notion of equivalence. The equivalence considered here is a modified

Jie Fu and Bert Tanner are with the Department of Mechanical Engineering, University of Delaware, Newark DE 19716. E-mail: {jiefu,btanner}@udel.edu.

Jeff Heinz is with the Department of Linguistics and Cognitive Science, University of Delaware, Newark DE 19716. E-mail: heinz@udel.edu.

This work is supported by NSF under the grant with award #1035577.

version of game equivalence in [16], and leads to guarantees that even if the true model of the environment is never found, the controllers built based on the equivalent model are effective in terms of satisfying the system specification. The result is a learning framework that combines concepts of GI with ideas from action model learning [17], capable of interfacing with a range of available symbolic control methods.

II. Languages, automata and games

Let Σ denote a fixed, finite alphabet, and Σ^*, Σ^+, Σ^ω be sequences over this alphabet of any finite length, of any finite length greater than 0, and of infinite length, respectively. The *empty string* is denoted λ, and the *length of string* w is denoted $|w|$. A regular (resp. ω-regular) *language* L is a subset of Σ^* (resp. Σ^ω). Given an ω-word w, $\mathsf{Occ}(w)$ denotes the set of symbols occurring in w and $\mathsf{Inf}(w)$ is the set of symbols occurring infinitely often in w. For a finite word $w \in \Sigma^*$, $\mathsf{last}(w)$ denotes the last symbol of w. Given a tuple $\boldsymbol{s} = (s_1, \ldots, s_N) \in S_1 \times S_2 \ldots \times S_N$, a projection operator is defined as $\pi_i(\boldsymbol{s}) = s_i$, for $0 \leq i \leq N$. For a set of tuples S, we write $\pi_i(S) = \bigcup_{\boldsymbol{s} \in S} \{\pi_i(\boldsymbol{s})\}$, and for a sequence of tuples $w = \boldsymbol{s}_1 \boldsymbol{s}_2 \cdots$ we apply the projection element-wise: $\pi_i(w) = \pi_i(\boldsymbol{s}_1)\pi_i(\boldsymbol{s}_2)\cdots$.

A semiautomaton (SA) deterministic in transitions is a tuple $A = \langle Q, \Sigma, T \rangle$ where Q is the finite set of states, Σ is the alphabet and the transition function is $T : Q \times \Sigma \to Q$. The assignment $T(q_1, \sigma) = q_2$ is also written $q_1 \xrightarrow{\sigma} q_2$, and is expanded recursively, i.e., $T(q_1, \lambda) = q_1$ and $T(q_1, uw) = T(T(q_1, u), w)$, for $u, w \in \Sigma^*$. We write $T(q, \sigma) \downarrow$ to express that the function T is defined for (q, σ), and $T(q, \sigma) \uparrow$ otherwise. A word $w \in \Sigma^*$ (resp. Σ^ω) is *admissible from* $q \in Q$ iff $T(q, w) \downarrow$. A *run* of A on a word (resp. ω-word) $w = w(0)w(1) \ldots \in \Sigma^*$ (resp. Σ^ω) is a finite (resp. infinite) sequence of states $\rho = \rho(0)\rho(1)\rho(2) \ldots \in Q^*$ (resp. Q^ω) such that $\rho(i+1) = T(\rho(i), w(i))$, $i \geq 0$. In this context, all SAs are deterministic in transition. The transition function can be made *total* by adding a state sink such that for all $q \in Q$, and for any $\sigma \in \Sigma$ for which $T(q, \sigma) \uparrow$, $T(q, \sigma) = \mathsf{sink}$. For all $\sigma \in \Sigma$, let $T(\mathsf{sink}, \sigma) = \mathsf{sink}$. An SA is total when its transition function is.

A deterministic automaton is a quintuple $\mathcal{A} = \langle Q, \Sigma, T, I, \mathsf{Acc} \rangle$ where $\langle Q, \Sigma, T \rangle$ is an SA, I is the *initial state*, and Acc is the *acceptance component*. An Acc can give rise to: 1) a deterministic finite state automaton (DFA), for which $\mathsf{Acc} = F \subseteq Q$, and $\mathcal{A}$ accepts $w \in \Sigma^*$ iff the run $\rho \in Q^*$ on w satisfies $\rho(0) \in I$ and $\mathsf{last}(\rho) \in F$; or 2) a deterministic Büchi automaton (DBA), for which $\mathsf{Acc} = F \subseteq Q$, and $\mathcal{A}$ accepts $w \in \Sigma^\omega$ iff the run $\rho \in Q^\omega$ on w satisfies $\rho(0) \in I$ and $\mathsf{Inf}(\rho) \cap F \neq \emptyset$. The set of words accepted by $\mathcal{A}$ is its *language* denoted $L(\mathcal{A})$. Given an automaton $\mathcal{A}$, we assume that, unless otherwise specified, A is the semiautomaton *obtained* from $\mathcal{A}$ by removing the markings on the initial and final states from $\mathcal{A}$.

Definition 1 (Two-player turn-based zero-sum game [11]): A two-player zero-sum turn-based game is a tuple $\mathcal{G} = \langle V_1 \cup V_2,\ \Sigma_1 \cup \Sigma_2, T, I, F \rangle$, where V_i is the set of states where player i moves, Σ_i is the set of actions for player i with $V_1 \cap V_2 = \Sigma_1 \cap \Sigma_2 = \emptyset$ and $V = V_1 \cup V_2$; $T : V_i \times \Sigma_i \to V_j$ is the transition function, $(i, j) \in \{(1, 2), (2, 1)\}$; I is the set of initial states, and $F \subseteq V_1 \cup V_2$ is the *winning condition*. If $\mathcal{G}$ is a reachability (or safety) game, then a run ρ is winning for player 1 if $\mathsf{last}(\rho) \in F$ (or $\mathsf{Occ}(\rho) \subseteq F$); if it is a Büchi game, then the condition is $\mathsf{Inf}(\rho) \cap F \neq \emptyset$. In any case the condition does not hold, player 2 wins.

A *strategy* for player i is a function $\mathsf{S}_i : V^* V_i \to \Sigma_i$ which maps a finite run ρ into an action $\mathsf{S}_i(\rho) \in \Sigma_i$ to be taken by player i. Player i *follows* strategy S_i if she always plays the action indicated by S_i. A strategy WS is a *winning* for player i if every run in $\mathcal{G}$ that starts in some $v \in V$ with player i following WS_i, is winning for player i. The *winning set* of player i, denoted $\mathsf{Win}_i \subseteq V$ is a collection of states, from each of which player i has a winning strategy.

III. Problem formulation

A. Problem statement

In this paper, we solve the following problem:

Problem 1: A controllable agent aims to accomplish a task specified in a linear temporal logic (LTL) formula, while interacting with an *unknown yet rule-governed* environment. The agent has some limited *prior* knowledge of the environment (as in [14], [15]), and must use this knowledge to identify a model of environment's behavior in the interaction in order to adapt its own, for the purpose of accomplishing its task whenever such an outcome is possible.

B. System interactions as a game

We think of the agent as player 1, its environment as player 2, and their interaction as a game.

The set of atomic propositions $\mathcal{AP}$ is used to describe their interactions. Given $\mathcal{AP}$, let the set of *world states* over $\mathcal{AP}$ be $\mathcal{C} = \{c = \ell_1 \wedge \ell_2 \ldots \wedge \ell_n \mid (\exists \alpha \in \mathcal{AP})[\ell_i = \alpha \vee \ell_i = \neg\alpha]\}$ and for any $c \in \mathcal{C}$, a proposition in $\mathcal{AP}$ appears *at most once*.

Discrete abstraction methods existing, we assume both players are modeled as labeled transition systems: $A_i = \langle Q_i, \Sigma_i, T_i, \mathcal{AP}_i, \mathsf{LB}_i \rangle$, $i = 1, 2$ in which $\langle Q_i, \Sigma_i, T_i \rangle$ is an SA; $\mathcal{AP}_i$ is a subset of atomic propositions $\mathcal{AP}$ which can be modified by player i's actions and $\mathcal{AP} = \mathcal{AP}_1 \cup \mathcal{AP}_2$; and $\mathsf{LB}_i : Q_i \to \mathcal{C}$ is the labeling function: for any $q \in Q_i$, let $\mathsf{LB}_i(q)$ be a world state over $\mathcal{AP}_i$.

We assume an action $\sigma \in \Sigma_i$ has conditional effects:

1) $\textsc{Pre}(\sigma) \in \mathcal{C}$ is the sentence that needs to be satisfied for the player to initiate action σ, and
2) $\textsc{Post}(\sigma) \in \mathcal{C}$ is the sentence that is satisfied when action σ is completed.

Given $c \in \mathcal{C}$, if $c \implies \textsc{Pre}(\sigma)$ where $\implies$ is the logical connective for implication, then the *effect of action σ on c*, denoted $\sigma(c) \in \mathcal{C}$ is the *unique* world state after performing σ when the world state is c.

In our setting players act in alternation. However, with the inclusion of the dummy action "idle," they may not necessarily play by turns. The following product captures all their possible interactions, and with the understanding

that $\mathcal{AP}$ is shared by both players, $\mathcal{AP}$ is omitted from the product.

Definition 2 (Turn-based product): Given two players $A_1 = \langle Q_1, \Sigma_1, T_1, \mathsf{LB}_1\rangle$ and $A_2 = \langle Q_2, \Sigma_2, T_2, \mathsf{LB}_2\rangle$, their turn-based product $P = \langle Q, \Sigma, \delta, \mathsf{LB}\rangle$ is an SA denoted $A_1 \circ A_2$, defined as follows:

- Q $= Q_1 \times Q_2 \times \{\mathbf{1}, \mathbf{0}\}$, where $\mathbf{t} \in \{\mathbf{0}, \mathbf{1}\}$ marks whose turn it is: $\mathbf{1}$ for player 1 and $\mathbf{0}$ for player 2.
- Σ $= \Sigma_1 \cup \Sigma_2$ is the alphabet.
- LB $: Q \to \mathcal{C}$ is the labeling function, defined by
$$\mathsf{LB}((q_1, q_2, \mathbf{t})) = \mathsf{LB}_1(q_1) \wedge \mathsf{LB}_2(q_2) \ ,$$
which maps a pair of player states into a world state corresponding to this pair.
- δ is the deterministic transition relation: $\delta\big((q_1, q_2, \mathbf{1}), \sigma\big) = (q_1', q_2, \mathbf{0})$ if $[T_1(q_1, \sigma) = q_1'] \wedge [\mathsf{LB}((q_1, q_2, \mathbf{1})) \implies \textsc{Pre}(\sigma)]$; $\delta\big((q_1, q_2, \mathbf{0}), \sigma\big) = (q_1, q_2', \mathbf{1})$ if $[T_2(q_2, \sigma) = q_2'] \wedge [\mathsf{LB}((q_1, q_2, \mathbf{0})) \implies \textsc{Pre}(\sigma)]$.

The objective of player 1 is represented by a language over the set of world states $\mathcal{C}$ accepted by a *total* automaton $\mathcal{A}_s = \langle S, \mathcal{C}, T_s, I_s, F_s\rangle$, where $\mathsf{sink} \in S$. The objective is a reachability (resp. Büchi) objective if $\mathcal{A}_s$ is a DFA (resp. DBA). Together, P and $\mathcal{A}_s$ define a game:

Definition 3 (Two-player turn-based game automaton): Given the turn-based product $P = \langle Q, \Sigma, \delta, \mathsf{LB}\rangle$ and the task specification $\mathcal{A}_s = \langle S, \mathcal{C}, T_s, I_s, F_s\rangle$, a two-player turn-based game automaton is constructed as a special product of P and $\mathcal{A}_s$, denoted $\mathcal{G} = P \ltimes \mathcal{A}_s = (A_1 \circ A_2) \ltimes \mathcal{A}_s = \langle V = V_1 \cup V_2, \Sigma_1 \cup \Sigma_2, T, I, F\rangle$, where

- V_1 $\subseteq \{(q, s) \mid q = (q_1, q_2, \mathbf{1}) \in Q \wedge s \in S\}$ and
- V_2 $\subseteq \{(q, s) \mid q = (q_1, q_2, \mathbf{0}) \in Q \wedge s \in S\}$
- T $: V \times \Sigma \to V$ such that $T((q, s), \sigma) = (q', s')$ iff $\delta(q, \sigma) = q'$, and $T_s(s, c) = s'$ with $c = \mathsf{LB}(q')$.
- I $= \{\, (q, s) \in V \mid s = T_s(I_s, \mathsf{LB}(q)) \,\}$ is the set of possible initial game states.
- F $= \{\, (q, s) \in V \mid s \in F_s \,\}$ is the winning condition.

With a slight abuse of notation, the labeling function in $\mathcal{G}$ is defined as $\mathsf{LB}(v) = \mathsf{LB}(\pi_1(v))$ where $\pi_1(v) \in Q$ and expanded as $\mathsf{LB}(\rho_1\rho_2) = \mathsf{LB}(\rho_1)\mathsf{LB}(\rho_2)$, for $\rho_1, \rho_2 \in V^*$.

For a fixed initial state $v_0 \in I$, when $\mathcal{A}_s$ is a DFA, $(\mathcal{G}, v_0)$ is a reachability game. When $\mathcal{A}_s$ is a DBA, $(\mathcal{G}, v_0)$ is a Büchi game. The runs in $(\mathcal{G}, v_0)$ and $\mathcal{A}_s$ are related as follows:

$$\begin{array}{lccccc} \rho_{\mathcal{G}}: & (q^{(0)}, s^{(0)}) & \xrightarrow{\sigma_1} & (q^{(1)}, s^{(1)}) & \ldots \\ \rho_s, \mathcal{A}_s: & I_s \xrightarrow{\mathsf{LB}(q^{(0)})} s^{(0)} & & \xrightarrow{\mathsf{LB}(q^{(1)})} s^{(1)} & \ldots \end{array} \tag{1}$$

where $\mathsf{LB}(q^{(1)}) = \sigma_1(\mathsf{LB}(q^{(0)}))$.

Player 1 winning in $(\mathcal{G}, v_0)$ means that the task specification encoded in $\mathcal{A}_s$ is satisfied:

Proposition 1: For any winning run $\rho \in V^*$(or V^ω) of player 1 in $\mathcal{G}$, $\mathsf{LB}(\rho) \in \mathcal{C}^*$(or $\mathcal{C}^\omega$) is accepted by $\mathcal{A}_s$.

Proof: Since ρ is winning for player 1, in a reachability (resp. Büchi) game, $\mathsf{last}(\rho) \in F$ (resp. $\mathsf{Inf}(\rho) \cap F \neq \emptyset$). Projecting ρ on the state set S of $\mathcal{A}_s$, we obtain $\mathsf{last}(\pi_2(\rho)) \in \pi_2(F) \subseteq F_s$ (resp. $\mathsf{Inf}(\pi_2(\rho)) \cap F_s \neq \emptyset$). Since the run in $\mathcal{A}_s$ corresponding to ρ is $\rho_s = I_s\pi_2(\rho)$ by (1), we have $\mathsf{last}(\rho_s) \in F_s$ (resp. $\mathsf{Inf}(\rho_s) \cap F_s \neq \emptyset$) and thus the word generating ρ_s, which is $\mathsf{LB}(\rho)$, is accepted in $\mathcal{A}_s$ by the definition of acceptance component. ∎

The turn-based product and the game automaton can be constructed in polynomial time in the size of their factors. The winning strategy of player 1, if it exists, becomes the controller that ensures that the task specification is satisfied, *irrespectively* of the changes in the environment. For reachability and Büchi games, WS_1, if exists, can be computed with methods in [11], [15], in linear (for reachability) and polynomial (for Büchi) time in the size of the game.

IV. Incorporating grammatical inference

When player 1 does not have complete knowledge of her opponent, and as a result of the game, the integration of GI as a learning mechanism in a setting where the game is repeated sufficiently many times, can eventually give player 1 a winning strategy.

A. Preliminaries: Grammatical Inference

A *positive presentation* ϕ of a language L is a total function $\phi : \mathbb{N} \to L \cup \{\#\}$ such that for every $w \in L$, there exists $n \in \mathbb{N}$ such that $\phi(n) = w$ [18]. Here $\#$ denotes a pause, a moment in time when no information is forthcoming. A presentation ϕ can also be understood as an infinite sequence $\phi(0)\phi(1)\cdots$ containing every element of L, interspersed with pauses. Let $\phi[i]$ denote the finite sequence $\phi(0)\phi(1)\ldots\phi(i)$.

Grammars are finite descriptions of potentially infinite languages. The language of grammar G is $L(G)$. A *learner* (*learning algorithm, or grammatical inference machine (GIM)*) is a program that takes the first i elements of a presentation, i.e. $\phi[i]$, and outputs a grammar G, written $\mathsf{GIM}(\phi[i]) = G$. The grammar outputted by GIM is the learner's *hypothesis* of the language. A learner GIM *identifies in the limit from positive presentations* a class of languages $\mathcal{L}$ if for all $L \in \mathcal{L}$, and for all presentations ϕ of L, there exists a $n \in \mathbb{N}$ such that for all $m \geq n$, GIM outputs a grammar $\mathsf{GIM}(\phi[m]) = G$, and $L(G) = L$ [19].

B. *Learning* the *game, or* a *game?*

Consider the set of finite prefixes of all possible behaviors of two players (sequences of interleaving actions) in $(\mathcal{G}, v_0)$ as a language, denoted $L(\mathcal{G}, v_0) = L(\langle V, \Sigma, T, v_0, V\rangle)$. The finite prefixes of the behavior of player $i \in \{1, 2\}$ is the projection of $L(\mathcal{G}, v_0)$ on Σ_i, denoted $L_i(\mathcal{G}, v_0)$. Due to the existence of constraints that one player forces on another through their interaction, for any $v_0 \in I$, $i = 1, 2$, $L_i(\mathcal{G}, v_0) \subseteq L(\mathcal{A}_i)$, where $\mathcal{A}_i = \langle Q_i, \Sigma_i, T_i, \pi_i(v_0), Q_i\rangle$ is the DFA obtained from SA A_i by assigning $\pi_i(v_0)$ as the initial, and all states final.

We assume the following condition for the result to apply.

Assumption 1: $L_2(\mathcal{G}, v_0)$ belongs to a class of languages identifiable in the limit from positive presentations, and player 1 has correct prior knowledge of the class of languages to which $L_2(\mathcal{G}, v_0)$ belongs.

Knowledge of the class of languages of $L_2(\mathcal{G}, v_0)$ is *required* and important in reducing the size of hypothesis

space. Such knowledge can be obtained by knowing some restrictions on the dynamics of, or the strategy that can be used by the adversary, and allows us to characterize the class of the languages modeling her behavior.

By playing the game repeatedly, a positive presentation of $L(\mathcal{G}, v_0)$ can be obtained: let $\phi(0) = \lambda$, and suppose that after move $i = 1, \ldots, n$, the obtained initial sequence of ϕ is $\phi[i]$, with the understanding that the move index i counts from the very first game, until the current move in the latest game. If move $i+1$ is the first in one of the games in the sequence, and player k plays $\sigma \in \Sigma_k$ then $\phi(i+1) = \sigma$; otherwise $\phi(i+1) = \phi(i)\sigma$. The projection of a presentation ϕ on the alphabet of player 2 is denoted ϕ_2.

Let GIM be an algorithm that identifies a class of languages $\mathcal{L}$ in the limit from positive presentations and let $L_2(\mathcal{G}, v_0) \in \mathcal{L}$. By definition [1], for any positive presentation ϕ of $L_2(\mathcal{G}, v_0)$, there exists $n \in \mathbb{N}$ such that $\forall m \geq n$, $L(\mathsf{GIM}(\phi_2[m])) = L_2(\mathcal{G}, v_0)$, where $\mathsf{GIM}(\phi_2[m])$ is the grammar produced by the learning algorithm.

However, even if $L_2(\mathcal{G}, v_0)$ is identified by GIM, the language of the game remains elusive, since the way these two players interfere with the available actions of each is still unknown. Moreover, due to this interference, $L_i(\mathcal{G}, v_0) \subseteq L(\mathcal{A}_i)$, for $i = 1, 2$. Thus, even though there exists more that one DFA $\mathcal{B}$ such that $L(\mathcal{B}) = L(\mathsf{GIM}(\phi_2[m]))$, one cannot just construct *any* DFA $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathsf{GIM}(\phi_2[m]))$ and expect the game automaton to be $(A_1 \circ B) \ltimes \mathcal{A}_s$ (B is the SA obtained from $\mathcal{B}$), since 1) the set of player 2 behaviors captured in B is a subset of that of A_2 and consequently 2) the labeling function in A_2 cannot be computed from B.

Hence, we instead use GIM to obtain a hypothesis of the game being played. This hypothesis converges to an *equivalent* one, which may not have the same transitions and states as the actual game that unfolds (cf. [15]). Nonetheless, it serves just as well: the winning strategy computed using the hypothesis eventually converges to one that is *isomorphic to* the true winning strategy for player 1 in the actual game.

C. Equivalence in games

Through the concept of bisimulation in transition systems we establish the equivalence between two games.

Definition 4: [20] A *bisimulation* of two transition systems $P = \langle Q, \Sigma, \delta, \mathsf{LB}\rangle$ and $P' = \langle Q', \Sigma, \delta', \mathsf{LB}'\rangle$ is a binary relation $\mathfrak{R} \subseteq Q \times Q'$ that whenever $(q, q') \in \mathfrak{R}$ and $\sigma \in \Sigma$, the following conditions hold:

(i) $\mathsf{LB}(q) = \mathsf{LB}'(q')$.
(ii) if $\delta(q, \sigma) = p$, then $\delta'(q', \sigma) = p'$ for some $p' \in Q'$ such that $(p, p') \in \mathfrak{R}$.
(iii) if $\delta'(q', \sigma) = p'$, then $\delta(q, \sigma) = p$ for some $p \in Q$ such that $(p, p') \in \mathfrak{R}$.

We write $P \simeq P'$ if P and P' are bisimilar. For designated initial states q_0, q_0', we say (P, q_0) is bisimilar to (P', q_0') and write $(P, q_0) \simeq (P', q_0')$ if and only if after trimming all states inaccessible from q_0 and q_0', $P \simeq P'$ and $(q_0, q_0') \in \mathfrak{R}$.

The following definition is adapted from [16].

Definition 5 (Equivalence between games): Two games $(\mathcal{G}, v_0) = \langle V, \Sigma, T, v_0, F\rangle$ and $(\mathcal{G}', v_0') = \langle V', \Sigma, T', v_0', F'\rangle$ are *equivalent* if there exist two functions $r : V^* \to V'^*$ and $r' : V'^* \to V^*$ such that given a winning strategy of player 1 in $(\mathcal{G}, v_0)$, $\mathsf{WS}_1 : V^* \to \Sigma_1$, the strategy $\mathsf{WS}_1' : V'^* \to \Sigma_1$ defined by $\forall \rho' = v_0' v_1' \ldots v_n' \in V'^*$, $\mathsf{WS}_1'(\rho') = \mathsf{WS}_1(r'(\rho'))$ is winning for player 1 in $(\mathcal{G}', v_0')$ and vice versa.

Proposition 2: If (P, q_0) and (P', q_0) are bisimilar, then the games $(\mathcal{G}, v_0) = (P, q_0) \ltimes \mathcal{A}_s$ and $(\mathcal{G}', v_0') = (P', q_0') \ltimes \mathcal{A}_s$ are equivalent, for *any* deterministic objective automaton $\mathcal{A}_s = \langle S, \mathcal{C}, T, I_s, F_s\rangle$.

Proof: Define r, r' in Definition 5 using complete induction: for initial states $v_0 = (q_0, s_0)$ and $v_0' = (q_0', s_0)$ where $s_0 = T_s(I_s, \mathsf{LB}(q_0)) = T_s(I_s, \mathsf{LB}'(q_0'))$ since $\mathsf{LB}(q_0) = \mathsf{LB}'(q_0')$, let $r(v_0) = v_0'$ and $r'(v_0') = v_0$, we have $\pi_2(v_0) = \pi_2(v_0')$ and $(\pi_1(v_0), \pi_1(v_0')) = (q_0, q_0') \in \mathfrak{R}$; then suppose r, r' are defined for finite runs $\rho = v_0 v_1 \ldots v_n$ and $\rho' = v_0' v_1' \ldots v_n'$ such that $r(\rho) = \rho'$, $r'(\rho') = \rho$ and for all $0 \leq i \leq n$, $\pi_2(v_i) = \pi_2(v_i') \in S$ and $(\pi_1(v_i), \pi_1(v_i')) \in \mathfrak{R}$.

Consider $\sigma \in \Sigma$ for which $T(v_n, \sigma) \downarrow$, and let $v_{n+1} = T(v_n, \sigma)$. Suppose $v_n = (q_n, s_n)$ and $v_n' = (q_n', s_n')$, by the assumption of r, r', we have $(q_n, q_n') \in \mathfrak{R}$ and $s_n' = s_n$. Since $\mathcal{A}_s$ is total and $\delta'(q_n', \sigma) \downarrow$ by bisimulation, $T'(v_n', \sigma) \downarrow$. The transitions in $\mathcal{G}$ and $\mathcal{G}'$ are related:

$$\begin{array}{rccc} \mathcal{G}: & (q_n, s_n) & \xrightarrow{\sigma} & (q_{n+1}, s_{n+1}) \\ & \updownarrow \mathfrak{R} & & \updownarrow \mathfrak{R} \\ \mathcal{G}': & (q_n', s_n) & \xrightarrow{\sigma} & (q_{n+1}', s_{n+1}') \end{array}$$

where $s_{n+1} = T_s(s_n, \mathsf{LB}(q_{n+1}))$ and $s_{n+1}' = T_s(s_n, \mathsf{LB}'(q_{n+1}'))$. Since q_{n+1}' is related to q_{n+1} through $\mathfrak{R}$, from $\mathsf{LB}(q_{n+1}) = \mathsf{LB}'(q_{n+1}')$ we must have $s_{n+1}' = s_{n+1}$.

Let $r(\rho v_{n+1}) = \rho' v_{n+1}'$ and $r'(\rho' v_{n+1}') = \rho v_{n+1}$ and inductively it follows that for two runs $\rho \in V^*$ and $\rho' \in V'^*$ such that $r(\rho) = \rho'$ and $r'(\rho') = \rho$, it holds

$$(\forall i : 0 \leq i < |\rho|)[\pi_2(v_i) = \pi_2(v_i') \wedge (\pi_1(v_i), \pi_1(v_i')) \in \mathfrak{R}] \ .$$

Now suppose $\mathsf{WS}_1 : V^* \to \Sigma_1$ is a winning strategy for player 1 in $(\mathcal{G}, v_0)$. For any run ρ produced by player 1 applying WS_1, let $r(\rho)$ be the run produced by player 1 applying WS_1' (Definition 5) and note that $\mathsf{LB}(\rho) = \mathsf{LB}'(r(\rho))$, where LB and LB′ are the labeling functions of $\mathcal{G}$ and $\mathcal{G}'$, respectively. Because of this latter equality between the images of the labeling functions, when ρ is winning for player 1, by Proposition 1 we can infer $\mathsf{LB}(\rho)$ is accepted by $\mathcal{A}_s$, and consequently $r(\rho)$ is winning for player 1 in $(\mathcal{G}', v_0')$ since $\mathsf{LB}'(r(\rho)) = \mathsf{LB}(\rho)$ is accepted by $\mathcal{A}_s$ as well. ■

Proposition 2 sets the theoretical foundation that allows us to compute a winning strategy for player 1 in a game *equivalent* to the original one when the latter is *unknown* but can be *learned* from positive presentations.

D. Learning an equivalent game from positive presentations

The learning module in our framework combines two learning processes that work in parallel: one aims to identify a transition system that keeps track of the updates of world states during the course of the game, and the other is a typical GIM. By combining these two we are able to compute a game equivalent to the *true* game in the sense of Definition 5.

We assume player 1 always knows whose turn it is (i.e. the Boolean value $\mathfrak{t} \in \{0,1\}$) and has full observation of the set of atomic propositions $\mathcal{AP}$, i.e., at any time instance during the game, player 1 knows the current evaluation of α for each $\alpha \in \mathcal{AP}$ whose value can be determined (either true or false). In the repeated game, the (move) index i counts from the very first game until the current move.

Definition 6 (World state transition system): During the game in which $(\mathcal{G}, v_0)$ being played repeatedly, the *world state transition system* constructed by player 1 at index n is $W(n) = \langle \mathcal{C} \times \{\mathbf{0},\mathbf{1}\}, \Sigma, T_w, (c_0, \mathfrak{t}_0)\rangle$, where $\mathcal{C} \times \{\mathbf{0},\mathbf{1}\}$ is a set of states and $(c_0, \mathfrak{t}_0)$ is the initial state;[1] $T_w : (\mathcal{C} \times \{\mathbf{0},\mathbf{1}\}) \times \Sigma \to \mathcal{C} \times \{\mathbf{0},\mathbf{1}\}$ is the transition relation defined based on the observations of player 1 as follows:

1) $\mathfrak{t} = \mathbf{1}$: $T_w((c,\mathbf{1}),\sigma) = (c',\mathbf{0})$ is defined if for $c \in \mathcal{C}$ with $c \implies \textsc{Pre}(\sigma)$, we have $c' = \sigma(c)$ is the world state that captures the effect of σ on world state c.
2) $\mathfrak{t} = \mathbf{0}$: $T_w((c,\mathbf{0}),\sigma) = (c',\mathbf{1})$ is defined if for $c \in \mathcal{C}$, after player 2 plays σ, the observed world state is c'.

In the course of the game $\mathcal{G}$, player 1 updates W as follows. Let the world state at index n be $c \in \mathcal{C}$. At $n+1$, suppose player 2 plays $\sigma \in \Sigma_2$ and the world state becomes c'. Then $W(n+1)$ is obtained from $W(n)$ incrementally by first adding state $(c',\mathbf{1})$ if it is not already included in the state set and then defining a transition $T_w((c,\mathbf{0}),\sigma) = (c',\mathbf{1})$ if not already existing. Outgoing transitions of $(c',\mathbf{1})$ are subsequently added according to the definition of T_w: for any $\sigma \in \Sigma_1$ such that $\textsc{Pre}(\sigma)$ is satisfied by c', we add a transition from $(c',\mathbf{1})$ to $(\sigma(c'),\mathbf{0})$ labeled σ.

The incremental construction of W is reminiscent of learning an *action model* with full observation [17] by treating the action of player 2 as the set of actions whose conditional effects have to be learned. The convergence of learning is guaranteed: informally, according to the turn-based product P, the set of world states that may actually be encountered during players' interaction is fixed. Once the set of states in $W(i)$ for some $i \in \mathbb{N}$ converges to a set that contains $\mathsf{LB}(Q)$, then one adds transitions with a known state set using the rules defined above. The convergence is reached when no more transition can be added.

Definition 7: Suppose that upon the initialization of the game $(\mathcal{G}, v_0)$ player 1 is at state I_1, $W(n) = \langle \mathcal{C} \times \{\mathbf{0},\mathbf{1}\}, \Sigma, T_w, (c_0,\mathfrak{t}_0)\rangle$ has been constructed at index n and a hypothesis of $L_2(\mathcal{G}, v_0)$ is given in the form of a grammar G by a GIM, for which one finds a DFA $\mathcal{B} = \langle Q_2^h, \Sigma_2, T_2^h, I_2^h, F_2^h\rangle$ such that $L(\mathcal{B}) = L(G)$. The *hypothesized turn-based product* is

$$\begin{aligned}&\mathrm{HP} = W(n) \times_s \mathcal{A}_1 \times_s \mathcal{B}\\ &= W(n) \times_s \langle Q_1, \Sigma_1, T_1, I_1, Q_1\rangle \times_s \mathcal{B} = \langle H, \Sigma, \delta', h_0, \mathsf{LB}'\rangle\end{aligned}$$

where $H = \mathcal{C} \times \{\mathbf{1},\mathbf{0}\} \times Q_1 \times Q_2^h$ is the state set and $h_0 = (c_0, \mathfrak{t}_0, I_1, I_2^h)$ is the initial state; the transition relation δ' is defined as follows: for $h = (c, \mathfrak{t}, q_1, q_2^h)$,

1) $\sigma \in \Sigma_1 \wedge \mathfrak{t} = \mathbf{1}$: $\delta'(h,\sigma) = (T_w((c,\mathfrak{t}),\sigma), T_1(q_1,\sigma), q_2^h)$;
2) $\sigma \in \Sigma_2 \wedge \mathfrak{t} = \mathbf{0}$: $\delta'(h,\sigma) = (T_w((c,\mathfrak{t}),\sigma), q_1, T_2^h(q_2^h,\sigma))$.

The labeling function $\mathsf{LB}' : H \to \mathcal{C}$ is defined such that for any $h = (c, \mathfrak{t}, q_1, q_2^h) \in H$, $\mathsf{LB}'(h) = \pi_1(h) = c$.

Theorem 1: Let GIM be a learning algorithm that identifies in the limit from positive presentations a class of regular languages $\mathcal{L}$. Suppose game $(\mathcal{G}, v_0)$ with $v_0 = (q_0, s_0)$ is such that $L_2(\mathcal{G}, v_0) \in \mathcal{L}$, and consider any positive presentation of $L_2(\mathcal{G}, v_0)$ denoted $\phi_2 : \mathbb{N} \to L_2(\mathcal{G}, v_0) \cup \{\#\}$. Let the algorithm Alg be defined by

$$\mathsf{Alg}(\phi_2[n]) \triangleq (W(n) \times_s \mathcal{A}_1 \times_s \mathcal{A}_2(\phi_2[n])) \ltimes \mathcal{A}_s \ ,$$

where $n \in \mathbb{N}$, $\mathcal{A}_2(\phi_2[n]) = \langle Q_2^h, \Sigma_2, T_2^h, I_2^h, F_2^h\rangle$ is a DFA that accepts the language generated by the grammar $\mathsf{GIM}(\phi_2[n])$, and $\mathcal{A}_1$ is obtained from A_1 by assigning $I_1 = \pi_1(q_0)$ as the initial state and all states final. Then there exists some index $N \in \mathbb{N}$, such that

1) $L(\mathsf{GIM}(\phi_2[N])) = L_2(\mathcal{G}, v_0)$;
2) $W(n) = W(N)$ for all $n \geq N$;
3) $\mathsf{Alg}(\phi_2[N])$ is game equivalent to $(\mathcal{G}, v_0)$.

Proof: It suffices to prove that at index $N \in \mathbb{N}$, the hypothesized turn-based product $W(N) \times_s \mathcal{A}_1 \times_s \mathcal{A}_2(\phi_2[N]) \simeq (P, q_0) = \langle Q, \Sigma, \delta, q_0, \mathsf{LB}\rangle$, because it follows from Proposition 2 that $\mathsf{Alg}(\phi_2[N])$ is equivalent to $(\mathcal{G}, v_0)$.

At index N, let the hypothesized turn-based product be $\mathrm{HP} = W(N) \times_s \mathcal{A}_1 \times_s \mathcal{A}_2(\phi_2[N]) = \langle H, \Sigma, \delta', h_0, \mathsf{LB}'\rangle$. Let the relation $\mathfrak{R} \subseteq Q \times H$ be defined as $(q_0, h_0) \in \mathfrak{R}$ with $(q,h) \in \mathfrak{R}$ whenever there exists $w \in \Sigma^+$ such that $\delta(q_0, w) = q$ and $\delta'(h_0, w) = h$. We show $\mathfrak{R}$ is a bisimulation:

First we show that if $(q,h) \in \mathfrak{R}$, then $\mathsf{LB}(q) = \mathsf{LB}'(h)$: given that $\mathsf{LB}'(h_0) = c_0$ is the world state at the initialization of game, $\mathsf{LB}'(h_0) = \mathsf{LB}(q_0) = c_0$ due to the uniqueness of the initial world state. For $(q,h) \in \mathfrak{R}$, we assume without loss of generality that there exists $\sigma \in \Sigma$ for which $\delta(q,\sigma) = q'$ and $\delta'(h,\sigma) = h'$. Since in a deterministic game, for the same world state c the world state resulting from applying action σ on c is unique, and given $\mathsf{LB}(q) = \mathsf{LB}'(h) = c$ we can infer $\mathsf{LB}(q') = \mathsf{LB}'(h') = \sigma(c) = c' \in \mathcal{C}$. Inductively, it follows that for any $(q,h) \in \mathfrak{R}$, $\mathsf{LB}(q) = \mathsf{LB}'(h)$.

Next we show that if $(q,h) \in \mathfrak{R}$, then for every σ such that $\delta(q,\sigma) = q' \downarrow$, there must exist $h' \in H$ such that $\delta'(h,\sigma) = h'$ and $(q',h') \in \mathfrak{R}$ and vice versa. So far, we have

$$\begin{aligned}(q,h) \in \mathfrak{R} \implies &\mathsf{LB}(q) = \mathsf{LB}'(h) = c\\ &\wedge (\exists w \in \Sigma^*)[\delta(q_0,w) = q \wedge \delta'(h_0,w) = h] \ .\end{aligned}$$

Consider any σ such that $\delta(q,\sigma) \downarrow$ and let $q' = \delta(q,\sigma) = \delta(q_0, w\sigma)$. In showing that $\delta'(h,\sigma) \downarrow$, two cases can arise:

Case 1: $\sigma \in \Sigma_1$. By definition of the turn-based product, and from the fact that σ is taken at state q of P, we can infer $\mathsf{LB}(q) \implies \textsc{Pre}(\sigma)$, which means in $W(N)$, $T_w((c,\mathbf{1}),\sigma) = (\sigma(c),\mathbf{0})$ is defined. Meanwhile as $\delta(q_0, w\sigma) \downarrow$, let u_1 be the projection of w on Σ_1, and note that $u_1\sigma \in L_1(\mathcal{G}, v_0) \subseteq L(\mathcal{A}_1)$ implies $T_1(I_1, u_1\sigma) \downarrow$. By Definition 7, we have $\delta'(h,\sigma) \downarrow$. Let $h' = \delta'(h_0, w\sigma) = \delta'(h,\sigma)$. Then $(q',h') \in \mathfrak{R}$ by the definition of $\mathfrak{R}$.

[1] By construction, not all states in $\mathcal{C} \times \{\mathbf{0},\mathbf{1}\}$ can be accessed in $W(n)$.

Case 2: $\sigma \in \Sigma_2$. Similarly to the previous case, since $\delta(q_0, w\sigma) \downarrow$, let u_2 be the projection of w on Σ_2, and note that $u_2\sigma \in L_2(\mathcal{G}, v_0) \subseteq L(\mathcal{A}_2)$. As $L(\mathcal{A}_2(\phi_2[N])) = L_2(\mathcal{G}, v_0)$ in the limit, it follows that $T_2^h(I_2^h, u_2\sigma) \downarrow$. For $T_w((c, \mathbf{0}), \sigma)$ to be defined in $W(N)$, player 1 must have observed player 2 taking action σ when the world state is c. In the true turn-based product, given $\mathsf{LB}(q) = c$, unless player 2 never plays σ at q (in which case we can safely assume $w\sigma \notin L(\mathcal{G}, v_0)$), there must exist a time index $k \leq N$ when the transition labeled σ from $(c, \mathbf{0})$ is added to $W(k)$. Now since $T_2^h(I_2^h, u_2\sigma) \downarrow$ and $T_w((c, \mathbf{0}), \sigma) \downarrow$, by construction we have that $h' = \delta'(h, \sigma) = \delta'(h_0, w\sigma) \downarrow$, and thus $(q', h') \in \mathfrak{R}$ by the definition of $\mathfrak{R}$.

Till now, we have shown that P is simulated by HP: any sequence of actions of player 1 and 2 in P can be matched with a sequence of actions in HP. For the other direction note that any sequence of actions in HP can also be matched by a sequence of actions in P, because observed behaviors originate from the true turn-based product P that captures all possible interactions. Having shown $W(N) \times_s \mathcal{A}_1 \times_s \mathcal{A}_2(\phi_2[N]) \simeq (P, q_0)$, Proposition 2 allows us to conclude that the game $\mathsf{Alg}(\phi_2[N])$ is equivalent to $(\mathcal{G}, v_0)$. ∎

We say Alg identifies *a game equivalent to* $(\mathcal{G}, v_0)$ in the limit from positive presentations of $L_2(\mathcal{G}, v_0)$. From Theorem 1 and Proposition 2, it follows that the winning strategy of player 1 computed using $\mathsf{Alg}(\phi_2[N])$ ensures the satisfaction of the task specification accepted by $\mathcal{A}_s$. The winning strategy computed in $\mathsf{Alg}(\phi_2[N])$ converges, through functions r, r', to the winning strategy for player 1 in the $(\mathcal{G}, v_0)$. There is no need to compute r and r' explicitly because for any finite run ρ in $\mathsf{Alg}(\phi_2[N])$, $\mathsf{WS}_1'(\rho')$ computed using $\mathsf{Alg}(\phi_2[N])$ is exactly the action given by $\mathsf{WS}_1(r'(\rho'))$ in $(\mathcal{G}, v_0)$.

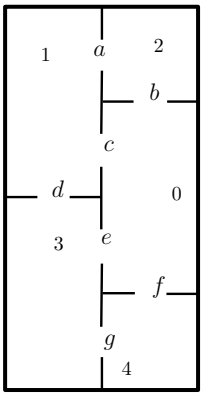
Until GIM converges, there can be no guarantee that an effective strategy can be found. However, since the output of GIM is always consistent with the history of observed environment behavior, the adaptive controller performs at least as good as any other synthesized one without making any inference. With observations accumulating, the adaptive controller is monotonically improving.

V. Case study

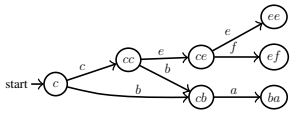
In which follows, we illustrate our method using an example: a robot (player 1) aims to visit rooms 1 through 4 in Fig. 1a, when the doors a, b, c, d, e, f, g are controlled by an adversary (player 2).

We assume player 2 adheres to the following rules: 1) doors b, c, e, f (around room 0) can be kept closed for at most two rounds, and the rest can be closed for at most one round.[2] 2) two doors closed consecutively, if not the same, must be adjacent to each other (doors are adjacent if connected via a wall; for example, doors b, c are adjacent to a.). Player 1 can either stay in her current room or move to an adjacent one, if the door connecting the two is open, but cannot stay in rooms $3, 4, 0$ for more than one round. Note that although in principle our method applies to cases where player 1 restricts the behavior of player 2, in this particular example this is not the case.[3]

As the first game starts, player 1 is informed that the language of player 2 is in the class of strictly 3-local languages [21] (see Appendix for characterization and available GIMs for this class of languages). Such information is inferred from the fact that player 2 has a finite memory of size 3. Figure 1b gives a graphical description of a fraction of the game automaton $(\mathcal{G}, v_0)$, which totally has 1214 states and 3917 transitions. The winning region for player 1 contains 996 states.



(a) A graphical depiction of the environment



(b) A fragment of A_2 where $I_2 = c$. Since player 2 cannot close b, c, e, f for more than two rounds, she needs to maintain a memory of size 2 keeping track of recently closed two doors, e.g. ab means doors a is closed in previous turn and the current closed door is b. Upon initialization, c means so far only door c has been closed.

Fig. 1: The environment and its abstraction.

Figure 3 shows how the learning algorithm converges after about 3895 turns (368 games). Convergence is quantified by measuring the ratio of the size (cardinality) of grammar $\mathsf{GIM}(\phi[n])$ over that of the grammar describing $L_2(\mathcal{G}, v_0)$; the latter has 121 factors of length 3 (see Appendix for background details). Interestingly, although it takes 368 games for the learning algorithm to converge, we observe that after 7 games player 1 only loses once (in the 51st game). This fact suggests that the controllers computed using the hypothesized games, even when those are not game-equivalent to the actual game, can still be effective. We compare this outcome with the case where player 1 has no capacity to learn, and replans in a way similar to [10] using a naive model for player 2: when player 1 observes a door is closed (open), she will assume the door will remain closed (open) until she observes it opening (closing) again. With player 2 exploiting every opportunity to prevail, player 1 achieves a win ratio of 27% when no learning is employed, compared to a ratio of 98% when GIM is used.

VI. Conclusion

The interaction of two discrete dynamical systems, where one of them needs to satisfy a specification without knowing the dynamics of the other on the outset, can be captured in

[2]Doors can be automatic sliding doors with different—designed—closing time spans.

[3]Perhaps that could happen if the robot were to remain at a certain door thus preventing it from closing, but this would not be particularly beneficial in terms of achieving its goal.

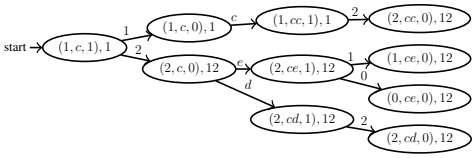


Fig. 2: A fraction of $(\mathcal{G}, v_0)$ where $v_0 = ((1, c, \mathbf{1}), 1)$. A state $((q_1, q_2, \mathfrak{t}), q_s)$ means the robot is in q_1, the recent consecutively closed (at most two) doors are q_2, $\mathfrak{t} = \mathbf{1}$ if player 1 is to make a move, otherwise player 2 is to make a move and the visited rooms are encoded in q_s, e.g. 12 means rooms $1, 2$ have been visited.

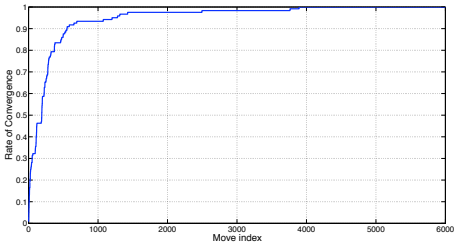


Fig. 3: Convergence of learning $L_2(\mathcal{G}, v_0)$: the ratio between the size of the grammar inferred by the GIM and that of $L_2(\mathcal{G}, v_0)$, in terms of number of moves made

a strictly competitive game of incomplete information. With the introduction of grammatical inference and by playing the game repeatedly, the agent tasked with satisfying the specification eventually does so after completing the missing information about its adversary. This completion takes the form of a transition system, based on which a game equivalent to the one actually being played can be constructed. Due to this equivalence, strategies computed on the hypothesized game, are just as effective as the true winning strategy computed on the game with complete information. Provided the right hypothesis of the class of unknown dynamics is made—meaning that the appropriate grammatical inference machine can be chosen—the hypothesized game converges to the one that is equivalent to the true game, in finitely many game rounds.

APPENDIX

A string u is a *factor* of string w iff $\exists x, y \in \Sigma^*$ such that $w = xuy$. If in addition u has length k, then u is a *k-factor* of w. The k-factor function $F_k : \Sigma^* \to 2^{\Sigma^{\leq k}}$ maps a word w to the set of k-factors within it if $|w| > k$; otherwise it maps w to the singleton $\{w\}$. We can extend F_k to a whole language, and write $F_k(L) := \bigcup_{w \in L} F_k(w)$. A language L is *Strictly k–Local* (SL_k) if there exists a finite set $G \subseteq F_k(\sharp \Sigma^* \sharp)$, such that $L = \{w \in \Sigma^* \mid F_k(\sharp w \sharp) \subseteq G\}$, where $\sharp$ is a special symbol indicating the beginning and ending of a string. The set G is the *grammar* that generates L.

A poly-time, incremental and set-driven learning algorithm for strictly k-local language is GIM defined by [21]: 1) $i = 0$: $\mathsf{GIM}(\phi[i]) := \emptyset$; 2) $\phi(i) = \#$: $\mathsf{GIM}(\phi[i]) := \mathsf{GIM}(\phi[i-1])$; 3) otherwise: $\mathsf{GIM}(\phi[i]) := \mathsf{GIM}(\phi[i-1]) \cup F_k(\sharp \phi(i) \sharp)$.

Example 1: Consider a strictly 2-local language $L = (\Sigma^* aa \Sigma^*)^{\complement} \cup (\Sigma^* ba \Sigma^*)^{\complement}$, for $\Sigma = \{a, b\}$, where $S^{\complement}$ be the complement of the set S with respect to Σ^*. In other words, L is the set of strings that don't have aa and ba factors. We have the grammar $G = F_k(L) = \{\sharp a, \sharp b, ab, bb, b\sharp, a\sharp\}$. Obviously, $aaa \notin L$ because $F_2(\sharp aaa \sharp) = \{\sharp a, aa, a\sharp\} \nsubseteq G$.

Learning proceeds as follows: given a positive presentation ϕ where $\phi(1) = ab$, $\phi(2) = bb$, $\phi(3) = a$, applying the learning algorithm $\mathsf{GIM}(\phi[1]) = F_k(\sharp ab \sharp) = \{\sharp a, ab, b\sharp\}$; $\mathsf{GIM}(\phi[2]) = \mathsf{GIM}(\phi[1]) \cup F_2(\sharp bb \sharp) = \{\sharp a, \sharp b, ab, bb, b\sharp\}$; $\mathsf{GIM}(\phi[3]) = \mathsf{GIM}(\phi[2]) \cup F_2(\sharp a \sharp) = G$. The learner converges after only having observed 3 strings.

REFERENCES

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

[2] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971 –984, july 2000.

[3] P. Tabuada, "Approximate simulation relations and finite abstractions of quantized control systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, A. Bemporad, A. Bicchi, and G. Buttazzo, Eds. Springer-Verlag, 2007, vol. 4416, pp. 529–542.

[4] ——, *Verification and control of hybrid systems: a symbolic approach*. Springer, 2009.

[5] H. Tanner, J. Fu, C. Rawal, J. Piovesan, and C. Abdallah, "Finite abstractions for hybrid systems with stable continuous dynamics," *Discrete Event Dynamic Systems*, vol. 22, pp. 83–99, 2012.

[6] G. E. Fainekos and H. Kress-Gazit, "Hybrid controllers for path planning: A temporal logic approach," in *Proceedings of the IEEE Conference on Decision and Control*, 2005, pp. 4885—4890.

[7] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, dec. 2009.

[8] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Hybrid Systems: Computation and Control*, K. H. Johansson and W. Yi, Eds. New York, NY, USA: ACM, 2010, pp. 101–110.

[9] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive high-level robot control," *IEEE Robotics and Automation Magazine*, pp. 65–74, September 2011.

[10] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal motion planning for hybrid systems in partially unknown environments," in *Hybrid Systems: Computation and Control*. New York, NY, USA: ACM, 2013, pp. 353–362.

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

[12] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kuwer, 1999.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[14] C. B. Yushan Chen, Jana Tumova, "LTL robot motion control based on automata learning of environmental dynamics," in *IEEE International Conference on Robotics and Automation*, Saint Paul, MN, USA, 2012.

[15] 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*, vol. 21, 2012, pp. 69–83.

[16] D. Berwanger and L. Kaiser, "Information tracking in games on graphs," *Journal of Logic, Language and Information*, vol. 19, no. 4, pp. 395–412, Oct. 2010.

[17] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[18] S. Jain, D. Osherson, J. S. Royer, and A. Sharma, *Systems That Learn: An Introduction to Learning Theory (Learning, Development and Conceptual Change)*, 2nd ed. The MIT Press, 1999.

[19] E. M. Gold, "Language identification in the limit," *Information and Control*, vol. 10, no. 5, pp. 447–474, 1967.

[20] C. Stirling, "The joys of bisimulation," in *Proceedings of 23rd International Symposium of Mathematical Foundations of Computer Science*, vol. 1450, 1998, pp. 142–151.

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