

# Adaptive Symbolic Control for Finite-state Transition Systems with Grammatical Inference

Jie Fu, Herbert G. Tanner, Jeffrey Heinz, and Jane Chandlee

**Abstract**—This note presents an approach that integrates elements from grammatical inference and game theory to address the problem of supervising finite-state transition systems operating in adversarial, partially known, rule-governed environments. The combined formulation produces controllers which guarantee that a transition system satisfies a task specification in the form of a logical formula, if and only (a) the true model of the environment is in the class of models inferable from positive data presentation (observations), (b) a characteristic sample of the environment’s behavior is observed, and (c) the task specification is satisfiable given the capabilities described by the abstractions of the system and its environment.

**Index Terms**—Hybrid systems, symbolic control, grammatical inference, algorithmic game theory.

## I. INTRODUCTION

This note shows how grammatical inference can be used in conjunction with standard game theoretic analysis to enable a transition system to satisfy a behavior specification while interacting with an adversarial, unknown, rule-governed environment. Transition systems such as the ones considered here may arise as discrete abstractions of hybrid dynamical systems. Specifics on the abstraction process itself can be found elsewhere (see [1]–[3]).

Conceptually similar problems have been studied in the context of reactive control [4]–[8], where system behavior is re-planned in real time based on information observed. A common underlying assumption is that the environment is *admissible*, that is, it cannot falsify the system’s specification. This assumption is not made here. Rather, we ask whether there is a method that reduces the problem for a system interacting with an *unknown* environment, into an instance where the environment is *known*, to allow the application of existing synthesis methods.

One way to answer this question is to learn, or identify a model for the environment. The learning paradigm adopted here is grammatical inference [9], [10], a methodology that identifies formal objects (e.g. languages) through presentation of examples of elements (e.g. strings in the language) with or without a teacher and/or oracle. With some prior knowledge about the object to be learned—narrowing down the search usually helps—correct identification occurs once a sufficient number of examples of the object’s features (the *characteristic sample*) is observed. In model diagnosis [11] and action model learning [12], a system is identified based on the observed behaviors and the predicated model for the system. It thus seems plausible that with some prior knowledge that helps to reduce our hypothesis space of the true system, one can start with an assumed model for the system and apply grammatical inference to identify the complete true system from observations.

Reinforcement learning offers an alternative formal methodology for regulating a system’s behavior while interacting with unknown dynamics (see [13] for an application in a discrete event system (DES) context). There are fundamental differences between the concept of reinforcement learning (RL) and that of grammatical inference. In a

Jie Fu, and Herbert Tanner are with the Mechanical Engineering Department at the University of Delaware, Newark DE 19716. {jiefu, btanner}@udel.edu.

Jeffrey Heinz and Jane Chandlee are with the Department of Linguistics and Cognitive Science at the University of Delaware, Newark DE 19716. {heinz, janemc}@udel.edu.

This work is supported by NSF award #1035577. The authors thank Jim Rogers for his insightful comments.

Markov decision process (MDP) setting, where RL takes place, the uncertainty about the environment interaction is expressed probabilistically, which is natural when there is inherent stochasticity in the models of the system or the environment. But when dynamics are purely deterministic, stochasticity becomes a modeling artifact. Furthermore, what reinforcement learning does is to enable the system to learn *how to exercise control*; in contrast, grammatical inference informs the system about *how to learn its environment*. Whereas grammatical inference has some precedence in supervisory control of discrete event systems [14], it has not been used for the model identification. Rather, it has been employed as a tool for learning the supremal controllable sublanguage for given system specifications, and in a setting where queries and the use of negative data is allowed. However, even when nature is not adversarial, she is unlikely to respond to our queries.

In this note we express the interaction between the system and its environment as a two-player zero-sum game. This corresponds to the worst-case assumption typically made for the environment in DES theory. In this form, the problem differs structurally from those treated using reinforcement learning approaches: when classifying games [15], a MDP is a 1-player stochastic game, whereas the games of this note are 2-player deterministic games. The note shows that with the suggested introduction of grammatical inference, control synthesis is decoupled from learning, and the latter can be performed outside a probabilistic framework. While demonstrating the approach, we perform synthesis using game-theoretic tools but that different options (e.g. [16]) are also available.

The rest of this note is organized as follows. Section II introduces the technical background, the notation, and the models used. In Section III, we briefly formulate the control problem into a game, in which the winning strategy of one player becomes the controller. In Section IV we introduce a learner to identify asymptotically the abstract model of the unknown and adversarial environment, and then how this knowledge can be utilized in planning and control synthesis. Section V illustrates the whole approach through an example. Section VI concludes.

## II. PRELIMINARIES

### A. Languages, automata and games

Let  $\Sigma$  denote a fixed, finite alphabet, and  $\Sigma^*$ ,  $\Sigma^\omega$  be sequences over this alphabet, of any finite length, and of infinite length, respectively. The *empty string* is denoted  $\lambda$ , and the *length of string*  $w$  is denoted  $|w|$ . A *language*  $L$  is a subset of  $\Sigma^*$ . A grammar  $G$  for a language  $L$  is finite description of  $L$  via a well-defined function  $L(\cdot)$  such that  $L(G) = L$  (an example is given later). A string  $u$  is a prefix (suffix) of a string  $w$  if there exists a string  $v$  such that  $w = uv$  ( $w = vu$ ). The prefix (suffix) of length  $k$  of a string  $w$  is denoted  $\text{Pr}^k(w)$  ( $\text{Sf}^k(w)$ ), and the set of prefixes (suffixes) of a string  $w$  of length  $\leq k$ ,  $\text{Pr}^{\leq k}(w)$  ( $\text{Sf}^{\leq k}(w)$ ). Given an  $\omega$ -word  $w$ ,  $\text{Occ}(w)$  denotes the set of symbols occurring in  $w$  and  $\text{Inf}(w)$  those occurring infinitely often in  $w$ . Given a finite word  $w \in \Sigma^*$ ,  $\text{last}(w)$  denotes the last symbol of  $w$ .

A semiautomaton is a tuple  $A = \langle Q, \Sigma, T \rangle$ , where  $Q$  is a set of states (here assumed finite),  $\Sigma$  is the finite alphabet, and  $T : Q \times \Sigma \rightarrow Q$  is the transition function. The mapping from  $(q_1, \sigma)$  to  $q_2$  via  $T$  is also written as  $q_1 \xrightarrow{\sigma} q_2$ , and can be expanded recursively in the usual way. Here, semiautomata are assumed deterministic in transitions. If  $T(q, \sigma)$  is defined for a given  $(q, \sigma) \in Q \times \Sigma$ , we write  $T(q, \sigma) \downarrow$ . The *active event function*  $\Gamma_A : Q \rightarrow 2^\Sigma$  singles out the alphabet symbols triggering outgoing transitions from a given state of  $A$ , and is defined as  $\Gamma_A(q) := \{\sigma \in \Sigma \mid T(q, \sigma) \downarrow\}$ . A *run* in  $A$  on an input word (resp.  $\omega$ -word)  $w = w(0)w(1) \dots \in \Sigma^*$  (resp.  $\Sigma^\omega$ ) is a

finite (resp. infinite) sequence of states  $\rho = \rho(0)\rho(1)\rho(2)\dots \in Q^*$  (reps.  $\in Q^\omega$ ) such that  $\rho(i+1) = T(\rho(i), w(i))$ ,  $i \geq 0$ . Then we say that  $\rho$  is *generated by*  $w$ . A semiautomaton  $A$  is *total* if for any  $(q, \sigma) \in Q \times \Sigma$ ,  $T(q, \sigma)$  is defined. Any semiautomaton  $A$  can be made *total* by adding a non-final state sink such that in every state, all symbols that are not in  $\Gamma_A(q)$  trigger transitions to sink, and at sink all symbols in  $\Sigma$  produce self-loops.

We think of a deterministic automaton as a quintuple  $\mathcal{A} = \langle Q, \Sigma, T, I, \text{Acc} \rangle$  where  $\langle Q, \Sigma, T \rangle$  is a semiautomaton deterministic in transitions,  $I$  is the initial state, and  $\text{Acc}$  is the *acceptance component*. The semantics of an acceptance component is context-based: if  $\mathcal{A}$  is a deterministic finite state automaton (DFA) with  $\text{Acc} = F \subseteq Q$ , then  $\mathcal{A}$  accepts  $w \in \Sigma^*$  iff run  $\rho \in Q^*$  generated by  $w$  satisfies  $\rho(0) \in I$  and  $\text{last}(\rho) \in F$ ; if  $\mathcal{A}$  is a deterministic Büchi automaton (DBA) with  $\text{Acc} = F \subseteq Q$ , then  $\mathcal{A}$  accepts  $w \in \Sigma^\omega$  iff run  $\rho \in Q^\omega$  on  $w$  satisfies  $\rho(0) \in I$  and  $\text{Inf}(\rho) \cap F \neq \emptyset$ . Language  $L(\mathcal{A})$  is the set of words accepted by  $\mathcal{A}$ . In the context of this note, calligraphic uppercase letters are reserved for DFAs.

The automaton form of a two-player turn-based game [17] 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  $i$  plays, and  $\Sigma_i$  is the set of actions for player  $i$ . The sets of player states and actions are disjoint:  $V_1 \cap V_2 = \Sigma_1 \cap \Sigma_2 = \emptyset$ . The transition function is  $T : V_i \times \Sigma_i \rightarrow V_j$ , with  $I$  the set of initial game states, and  $F \subseteq V_1 \cup V_2$  the *winning condition*. If  $\mathcal{G}$  is a reachability (or safety) game, then a run is winning for player 1 if  $\text{last}(\rho) \in F$  (or  $\text{Occ}(\rho) \subseteq F$ ); if it is a Büchi game, then the condition is  $\text{Inf}(\rho) \cap F \neq \emptyset$ .

A *memoryless strategy*<sup>1</sup> for player  $i$  in game  $\mathcal{G}$  is a function  $S_i : V_i \rightarrow \Sigma_i$  such that for every pair  $(v, \sigma)$  of preimage and image, a transition is defined. Player  $i$  follows strategy  $S_i$  if at state  $v \in V_i$ , player  $i$  always plays  $S_i(v)$ . A strategy is a *winning strategy* for player  $i$ , denoted  $\text{WS}_i$ , if every run in  $\mathcal{G}$  in which player  $i$  follows  $\text{WS}_i$ , is winning for him. The *winning set* of player  $i$ , denoted  $\text{Win}_i \subseteq V$  is the set of states from which there exists a winning strategy for player  $i$ .

### B. Grammatical inference

A *positive presentation*  $\phi$  of a language  $L$  is a total function  $\phi : \mathbb{N} \rightarrow L \cup \{\#\}$  such that for every  $w \in L$ , there exists  $n \in \mathbb{N}$  such that  $\phi(n) = w$  [10]. A presentation  $\phi$  can also be understood as an infinite sequence  $\phi(0)\phi(1)\dots$  containing every element of  $L$ , interspersed with pauses (marked with the  $\#$  symbol), which are moments in time when no information is forthcoming. We will take initial finite subsequences  $\phi(0)\phi(1)\dots\phi(i)$  and denote them  $\phi[i]$  to mark the (time) step at which the particular subsequence of symbols is observed. The *content* of  $\phi[i]$ , written  $\text{content}(\phi[i])$ , is the set of elements of the sequence, less the pauses. A grammatical inference machine (GIM) GIM, or learner for short, *identifies in the limit from positive presentations* a class of languages  $\mathcal{L}$  if for all  $L \in \mathcal{L}$ , and for all presentations  $\phi$  of  $L$ , there exists a  $n \in \mathbb{N}$  such that for all  $m \geq n$ ,  $\text{GIM}(\phi[m])$  outputs a grammar  $G$ , and  $L(G) = L$  [18]. Let  $C$  be a finite subset of  $L$ ,  $C$  is a *characteristic sample* of  $L$  for GIM if for any positive presentation  $\phi$ , for all  $n \geq 0$ ,  $C \subseteq \text{content}(\phi[n])$  implies  $\text{GIM}(\phi[n]) = L$  [9].

For concreteness, this note introduces a GIM with respect to the class of Strictly 2-Local languages; however, many other formal language classes have been shown to be learnable [9], any of which could be used in the current setting. A string  $u$  is a *factor* of another string  $w$ , if and only  $\exists x, y \in \Sigma^*$  such that  $w = xuy$ . All such

strings  $u$  of length  $k$  are the *k-factors* of  $w$ . The *k-factor function*  $\text{factor}_k : \Sigma^* \rightarrow 2^{\Sigma^{\leq k}}$  maps a word  $w$  to the set of  $k$ -factors in it, if  $|w| > k$ ; otherwise  $w$  maps to itself. We can extend  $\text{factor}_k$  to a whole language, and write  $\text{factor}_k(L) := \bigcup_{w \in L} \text{factor}_k(w)$ . Let  $\#$  be a special symbol marking the beginning and the end of a string. A language  $L$  is *Strictly k-Local* ( $\text{SL}_k$ ) if there exists a finite set  $G \subseteq \text{factor}_k(\#\Sigma^*\#)$ , such that  $L(G) = \{w \in \Sigma^* \mid \text{factor}_k(\#w\#) \subseteq G\}$ . Machines generating strictly  $k$ -local languages model processes where the next event depends only on the previous  $k - 1$  events. Informally, learning can occur because every observed string reveals some of the elements of the grammar. Therefore, after enough strings from the language are observed, the grammar—being a finite set—“fills out.” A formal definition of a GIM which learns in this way is provided later in Definition 3. The  $\text{SL}_k$  languages are identifiable in the limit from positive presentations [19] with a poly-time iterative and set-driven learner [20].

### III. SYSTEM BEHAVIOR AS GAME PLAY

In the interaction between two dynamical systems (the agent and its environment)—the players in the game—the actions of one have conditional effects over the state of their composition, which we refer to as *the world*. The world can be described through a formal system over a set of atomic propositions  $\mathcal{AP}$  [12]. A *literal* is either an atomic proposition in  $\mathcal{AP}$ , or a negation of an atomic proposition. A *sentence* is a conjunction of literals, in which each atomic proposition appears at most once. The set of all sentences that can be formed in this way makes a set of *world states* denoted  $\mathcal{C}$ .

In our context, a dynamical system is expressed as a special type of a Kripke structure. This structure is a semiautomaton augmented with a labeling function that maps a state to a *sentence* which is true at that state. Thus, player  $i$  is modeled as a tuple  $A_i = \langle Q_i, \Sigma_i, T_i, \text{LB}_i \rangle$ , where  $\text{LB}_i : Q_i \rightarrow \mathcal{C}$ . The set  $\Sigma_i$  may contain the empty action  $\epsilon$ , in which a player simply gives up her turn. All transitions in  $A_1$  (the system) are controllable, whereas transitions in  $A_2$  (the environment) are uncontrollable. We assume that the alphabets of the two players are disjoint, i.e.,  $\Sigma_1 \cap \Sigma_2 = \emptyset$ .

The conditional effect of action  $\sigma \in \Sigma_i$  is captured by its pre- and post-conditions. The pre-condition  $\text{PRE}(\sigma) \in \mathcal{C}$  is a sentence that has to be true in order for  $\sigma$  to occur. The post-condition  $\text{POST}(\sigma) \in \mathcal{C}$  is a sentence that must be true when the action is completed. Whenever  $\sigma \in \Gamma_i(q)$ , we have  $\text{LB}_i(q) \implies \text{PRE}(\sigma)$ ; similarly, when we see a transition from  $q$  to  $q'$  on action  $\sigma$ , denoted  $q \xrightarrow{\sigma} q'$ , we infer that  $\text{LB}_i(q') \implies \text{POST}(\sigma)$ .

We capture how each player can interfere with the dynamics of the other, by means of an *interaction function*  $U_i : Q_i \times Q_j \rightarrow 2^{\Sigma_j}$ , in which a pair of states  $(q_i, q_j)$  maps to the set of actions that player  $j$  can no longer take:  $\{a \in \Gamma_j(q_j) \mid \text{LB}_i(q_i) \wedge \text{LB}_j(q_j) \implies \neg \text{PRE}(a)\}$ . Intuitively, the interaction happens at the level of the images of their labeling functions, with some literals in  $\text{LB}_i(q_i)$ , negating some literals in  $\text{LB}_j(q_j)$ ; by doing so they falsify the pre-condition of action  $a \in \Gamma_j(q_j)$ .

We assume that the objective of each player is given as a logic formula over  $\mathcal{AP}$ . By restricting this objective to either first-order logic, or a fragment of Linear Temporal Logic (LTL) [21], the objective’s formula can be equivalently expressed as a DFA or DBA, respectively. These automata are referred to as the *objective automata*  $\mathcal{A}_s = \langle Q_s, \mathcal{C}, T_s, I_s, F_s \rangle$  where  $T_s$  is total, i.e.,  $\text{sink} \in Q_s$  and for any  $(q_s, c) \in Q_s \times \mathcal{C}$ ,  $T_s(q_s, c) \downarrow$ . Two types of objectives can be considered: a) Reachability or safety<sup>2</sup> objectives, in which  $\mathcal{A}_s$  is a DFA, and b) Büchi objectives, in which  $\mathcal{A}_s$  is a DBA.

<sup>1</sup>For two-player zero-sum reachability and Büchi games with perfect information, there always exists a memoryless winning strategy for one of the players [17].

<sup>2</sup>A safety objective is a dual of a reachability objective [17].

Our game is constructed in a bottom-up fashion, through appropriate product operations on the models of the players and the objective automaton. The first product introduced in Definition 1 constructs the game *arena*, which captures all interactions between players.

**Definition 1 (Turn-based product):** Given two players  $A_i = \langle Q_i, \Sigma_i, T_i, \text{LB}_i \rangle$ ,  $i = 1, 2$ , their turn-based product  $P = \langle Q_p, \Sigma_1 \cup \Sigma_2, T_p, \text{LB} \rangle$ , denoted  $A_1 \circ A_2$ , is defined as follows:  $Q_p$  is the set of states  $Q_1 \times Q_2 \times \{0, 1\}$ , where the last component is a Boolean variable  $t$  denoting whose *turn* it is to play:  $t = 1$  for player 1,  $t = 0$  for player 2.  $T_p$  is the transition function with  $T_p((q_1, q_2, 1), \sigma) = (q'_1, q_2, 0)$  if  $q'_1 = T_1(q_1, \sigma)$  and  $\sigma \notin U_2(q_2, q_1)$ , and  $T_p((q_1, q_2, 0), \sigma) = (q_1, q'_2, 1)$  if  $q'_2 = T_2(q_2, \sigma)$  and  $\sigma \notin U_1(q_1, q_2)$ .  $\text{LB}$  is the labeling function defined as  $\text{LB}((q_1, q_2, t)) = \text{LB}_1(q_1) \wedge \text{LB}_2(q_2)$ .

The following product completes the game construction by incorporating the players' objectives.

**Definition 2 (Game automaton):** Given the turn-based product  $P = A_1 \circ A_2 = \langle Q_p, \Sigma, T_p, \text{LB} \rangle$  and the objective automaton  $\mathcal{A}_s = \langle Q_s, \mathcal{C}, T_s, I_s, F_s \rangle$ , a two-player turn-based game automaton  $\mathcal{G} = (A_1 \circ A_2) \times \mathcal{A}_s = \langle V, \Sigma, T, I, F \rangle$  is defined, where

- $V = V_1 \cup V_2$ , where  $V_1 \subseteq Q_1 \times Q_2 \times \{1\} \times Q_s$  is the set of states where player 1 moves, and  $V_2 \subseteq Q_1 \times Q_2 \times \{0\} \times Q_s$  is the set of states where player 2 moves.
- $\Sigma = \Sigma_i \cup \Sigma_j$ , where  $\Sigma_i$  is the set of actions of player  $i$ .
- $T$  is the transition relation defined as follows:  $T((q_1, q_2, t, q_s), \sigma) = (q'_1, q'_2, t', q'_s)$  is defined iff  $(q'_1, q'_2, t') = T_p((q_1, q_2, t), \sigma)$ , and  $q'_s = T_s(q_s, \text{LB}((q'_1, q'_2, t')))$ .
- $I = \{ (q_1, q_2, t, q_s) \in V \mid q_s = T_s(I_s, \text{LB}(q_1, q_2, t)) \}$  is the set of initial game states.
- $F = \{ (q_1, q_2, t, q_s) \in V \mid q_s \in F_s \}$  is the set of final states.

For a given state  $v = (q_1, q_2, t, q_s) \in V$ , a projection operator  $\pi_i$  is defined such that  $\pi_i(v)$  is the  $i$ -th component in the tuple  $v$ . The game automaton has at most  $|Q_p| \times |Q_s|$  states. The labeling function  $\text{LB}$  is first extended from  $P$  to  $\mathcal{G}$ : for each state  $v = (q_1, q_2, t, q_s) \in V$ , we define  $\text{LB}(v) = \text{LB}(q_1, q_2, t)$ ; then it is extended to runs in  $V^* \cup V^\omega$  in the usual way. An *initialized* game is a tuple  $(\mathcal{G}, v_0)$  where  $v_0 \in I$  is the initial state. If  $\mathcal{A}_s$  is a DFA, game  $\mathcal{G}$  is a reachability game. Run  $\rho$  is winning for player 1 in game  $\mathcal{G}$  if  $\text{last}(\rho) \in F$ . Projecting the run on  $Q_s$ , we see that  $\mathcal{A}_s$  also accepts  $\text{LB}(\rho)$  since  $\text{last}(\pi_4(\rho)) \in \pi_4(F)$ , which is a subset of  $F_s$ . If  $\mathcal{A}_s$  is a DBA, game  $\mathcal{G}$  is a Büchi game and a run  $\rho$  is winning for player 1 in  $\mathcal{G}$  if  $\text{Inf}(\rho) \cap F \neq \emptyset$  and  $\mathcal{A}_s$  accepts  $\text{LB}(\rho)$  since  $\text{Inf}(\pi_4(\rho)) \cap F_s \neq \emptyset$ .

With the game expressed as a finite state machine, one can use standard supervisory control techniques [16] to synthesize controllers for player 1. As an alternative, we compute a winning strategy  $\text{WS}_1 : \text{Win}_1 \cap V_1 \rightarrow \Sigma_1$  in  $\mathcal{G}$  (which can be a reachability or a Büchi game) based on game-theoretic solutions [17], [21]. Applying these algorithms to games with  $m$  transitions and  $n$  states, results in time complexity  $\mathcal{O}(m + n)$  if they are reachability games, and in time complexity  $\mathcal{O}(n(m + n))$  if they are Büchi games [17].

#### IV. INCORPORATING GRAMMATICAL INFERENCE

Section III established that for a given task, an agent which (a) has full knowledge of the dynamical environment it interacts with, and (b) is initialized at a state in  $\text{Win}_1 \cap I$ , affords a controller that ensures that its objective is met. This section relaxes the condition on full knowledge by incorporating a GIM. The GIM updates the agent's model of the unknown and rule-governed environment via

observations made in the process of playing the game repeatedly, each time from an initial condition randomly selected from all possible initial states. During this process, the agent adjusts its behavior based on the latest instantiation of its environment model. In the limit, the system refines its model to one that accurately expresses its adversary's behavior, and thus recovers the performance achievable when full knowledge is assumed.

The assumptions that allow the implementation of the proposed approach are the following: a) Player 1 cannot restrict player 2, i.e.,  $\forall (q_1, q_2) \in Q_1 \times Q_2, U_1(q_1, q_2) = \emptyset$ ; b) The model of player 2 is identifiable in the limit from positive presentations by a GIM; c) Player 1 has prior knowledge for selecting the correct GIM; and d) the observed behavior of player 2 suffices for a correct inference to be made, i.e., it contains a characteristic sample.

These assumptions are for the most part conservative, and are justified as follows. The first assumption suggests that player 1 has a disadvantage: although player 2 can interfere with the execution of the plans of player 1, the actions of player 1 have no effect on the actions player 2 can take. The second assumption requires positive presentations only, because an adversary is unlikely to respond to queries (cf. [14]). This assumption can be relaxed in other grammatical inference frameworks (for instance, less prior knowledge is required when presentations contain both positive and negative data). The third and fourth assumptions simply state conditions for a GIM to converge; they are reasonable, in the sense that it is not beneficial for an adversary to consistently withhold action solely for the purpose of privacy. Note that this set of assumptions does not require that the environment cannot falsify the task specification (cf. [6]). Here environmental actions, and poor—due to ignorance—response by the system, can falsify the specification during the learning process of a repeated game [22]. When this happens, the game restarts from a new initial condition, with the agent *retaining* the knowledge accumulated until then.

Given a game  $\mathcal{G}$ , let  $L(\mathcal{G})$  be the set of prefixes of all possible action sequences made by interleaving the actions of player 1 with those of player 2, and  $L_2(\mathcal{G})$  be the projection of  $L(\mathcal{G})$  on  $\Sigma_2$ . The behavior of player 2 is a language  $L_2(\mathcal{G}) \in \Sigma_2^*$ . Based on assumption b) made earlier in this section, for  $\phi_2$  of  $L_2$ , there exists a GIM and  $n \in \mathbb{N}$  such that for all  $m \geq n$ ,  $\text{GIM}(\phi_2[m]) = \text{GIM}(\phi_2[n])$ . In addition, the language of the grammar given by GIM is  $L_2(\mathcal{G})$ , i.e.,  $L(\text{GIM}(\phi_2[n])) = L_2(\mathcal{G})$ .

Let the presentation of language  $L(\mathcal{G})$  obtained in the repeated game to be  $\phi$ , define  $\phi(0) = \lambda$ , and denote  $\phi[i]$  the presentation obtained after move  $i = 1, \dots, n$ . Since games are repeated, the move index  $i$  counts from the first move in the very first game until the current move in the latest game. If move  $i + 1$  is the first in one of the repeated games and player  $k$  plays  $\sigma \in \Sigma_k$  then  $\phi(i + 1) = \sigma$ ; otherwise  $\phi(i + 1) = \phi(i)\sigma$ . The projection of  $\phi$  on the alphabet of player 2 is a positive presentation of  $L_2(\mathcal{G})$ , denoted  $\phi_2$ .

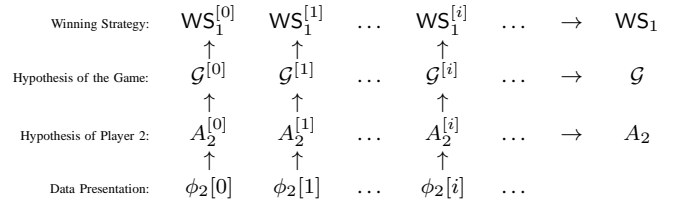


Fig. 1: Learning and planning with a grammatical inference module.

Figure 1 illustrates how identification in the limit proceeds. Through interactions with player 2, player 1 observes a finite initial segment of a positive presentation  $\phi_2[i]$  of  $L_2(\mathcal{G})$ , and uses the GIM to update a hypothesized model of player 2. Specifically, the output of  $\text{GIM}(\phi_2[i])$  becomes a DFA (see Appendix), which after

removing the initial state and the finality of final states, yields a semiautomaton  $A_2^{[i]}$ . The labeling function  $LB_2^{[i]}$  in  $A_2^{[i]}$  is defined as  $LB_2^{[i]} = \bigwedge_{\sigma \in \text{IN}(q)} \text{POST}(\sigma)$ , where  $\text{IN}(q) \triangleq \{\sigma \in \Sigma_2 \mid (\exists q' \in Q_2^{[i]})[T_2^{[i]}(q', \sigma) = q]\}$ ; this is the set of labels of incoming transitions of the state  $q$ . The computation of labeling function is of time complexity linear in the size of  $A_2^{[i]}$ . Given  $LB_2^{[i]}$ , the interacting function  $U_2(\cdot)$  is updated in linear time  $\mathcal{O}(|Q_1| \times |Q_2^{[i]}|)$ . Based on the interacting functions and the updated model for player 2, player 1 constructs a hypothesis (model for)  $\mathcal{G}^{[i]}$ , capturing her best guess of the game being played, and uses this model to compute  $WS_1^{[i]}$ , which converges to the true  $WS_1$  as  $A_2^{[i]}$  converges to the true  $A_2$ . Strategies  $WS_1^{[i]}$  for  $i < n$ , are the best responses for the system given the information it has so far, but having been devised based on incorrect hypotheses about the game being played, they cannot guarantee winning. There is no guaranteed upper bound on the number of games player 1 has to play before the learning process converges because one does not know at which point a characteristic sample of player 2's behavior is observed. However, as soon as this happens, convergence is guaranteed.

The game learning procedure is summarized in the following sequence of steps.

- 1) The game starts with initial state  $v_0 \in I$ ,  $i := 0$ , and the hypothesized game is  $\mathcal{G}^{[0]}$ .
- 2) At state  $v = (q_1, q_2, \mathbf{1}, q_s)$ , player 1 computes  $\text{Win}_1^{[i]}$  in  $\mathcal{G}^{[i]}$ . If  $v \in \text{Win}_1^{[i]}$ , a winning strategy  $WS_1^{[i]}$  exists in  $(\mathcal{G}^{[i]}, v)$ . Player 1 plays  $WS_1^{[i]}(v)$ , and proceeds to step 4. If  $v \notin \text{Win}_1^{[i]}$ , player 1 loses and jumps to step 3; if  $T(v, \sigma) \in F$ , player 1 wins and jumps to step 5.
- 3) With probability  $p$ , player 1 makes a move randomly selected from available moves at that time instance and jumps to step 4; or she jumps to step 5 with probability  $1 - p$ .
- 4) Player 2 makes a move. Player 1 observes the move, updates  $A_2^{[i]}$  to  $A_2^{[i+1]}$ , and  $\mathcal{G}^{[i]}$  to  $\mathcal{G}^{[i+1]}$ . Player 1 sets  $i := i + 1$  and goes to step 2.
- 5) The game is restarted at a random initial state  $v_0$  and If  $v_0 \notin \text{Win}_1^{[i]}$ , player 1 makes a random move and goes to step 4; otherwise, she jumps to step 2.

When player 1 finds herself out of her assumed winning set she can either quit and restart the game, or explore an action with some probability  $0 \leq p \leq 1$  and keep playing hoping that her opponent's response allows her to improve her hypothesis of the game. She has nothing to lose by trying to exploit her adversary's desire to win in order to extract information about her opponent's behavior.

Having no particular reason to choose otherwise, we define the utility or reward resulting from using a particular strategy as a binary function awarding 1 for a win and 0 for a loss. This reward is realized at the end of a game, and the regret  $r(S)$  that the system experiences for not following a strategy  $S$  in the process of learning, is the difference between the reward it gets while learning and the one it would have received if adhered to  $S$  [22]. The learning rule we introduce here is a *no regret* rule [22] in the sense that the average regret for not using a true winning strategy from the beginning—albeit there is no way to formulate it in advance based on available information—tends to zero as the game rounds increase. Specifically, if  $u(S)$  denotes the payoff (win: 1, lose: 0) of employing strategy  $S$ , realized at the end of the game, and  $WS_1$  is any true winning strategy for player 1, then  $\liminf_{i \rightarrow \infty} [u(WS_1^{[i]}) - u(WS_1)] \leq 0$ . This is due to the guaranteed convergence of the grammatical inference module in the limit, combined with the derivation of winning strategies on each hypothesized game  $\mathcal{G}^{[i]}$ .

## V. PLAYING FOR REAL

A robot (player 1) must visit all four rooms in the environment of Fig. 2, where doors connecting rooms are controlled by an adversary (player 2). The rules of the game are:

Rule 1: at each round, either no door opens (thus  $\epsilon \in \Sigma_2$ ), or one opens and another one closes;

Rule 2: doors closed must be opposite to each other, that is,  $\Sigma_2 = \{ad, ae, af, bf, ce, ef, \epsilon\}$  where  $ij$ , denotes doors  $i$  and  $j$  being closed.

Rule 1 makes  $L_2(\mathcal{G})$  a strictly 2-local (SL<sub>2</sub>) language [20], [23]. The graph of this language acceptor can be represented with a Myhill graph. Algorithms for deciding whether a language is strictly  $k$ -local and for which  $k$  it is can be found in [24].

The prior knowledge used in this game is the following: (i) Initially, player 1 is aware of Rule 2 only,<sup>3</sup> and (ii) player 1 starts the game knowing that the language of player 2 is strictly 2-local. Let  $\mathcal{AP} = \{\alpha_i : \text{robot in room } i\} \cup \{d_{ij} : \text{the door connecting rooms } i \text{ and } j \text{ is closed}\}$ . Without any other information about the doors' behavior, player 1 starts the game hoping that the doors will stay as they appear at the outset: all open. Player 1 plans with this in mind, modeling her own dynamics as  $A_1 = \langle Q_1, \Sigma_1, T_1, LB_1 \rangle$  where  $Q_1 = \{1, 2, 3, 4\}$ ,  $\Sigma_1 = Q_1 \times Q_1 \setminus \{(i, i) \mid i \in Q_1\}$ , and transition  $T_1(i, (i, j)) = j$  suggests movement from room  $i$  to  $j$ . We set  $LB_1(i) = \alpha_i$ . Player 1 models the opponent as  $A_2 = \langle Q_2, \Sigma_2, T_2, LB_2 \rangle$  where  $Q_2 = \Sigma_2 \setminus \epsilon$  and  $T_2(q_1, q_2) = q_2$  if  $q_2 \neq \epsilon$ , and  $T_2(q_1, \epsilon) = q_1$ . The labeling function  $LB_2$  is understood graphically, for example,  $LB_2(ad) = d_{12} \wedge d_{34}$  as  $a$  connects rooms 1, 2 and  $d$  connects 3, 4. The objective automaton is  $\mathcal{A}_s = \langle Q_s, \mathcal{C}, T_s, I_s, F_s \rangle$ , which is the canonical finite state automaton accepting the union of the shuffle ideals<sup>4</sup> of the permutations of string  $\alpha_1 \alpha_2 \alpha_3 \alpha_4$ . The interaction function  $U_2(dd', i)$  indicates the rooms player 1 cannot go to from room  $i$  due to doors  $d$  and  $d'$  being closed. The actions of player 1 do not inhibit that of player 2:  $U_1(q) = \emptyset, \forall q \in Q_1 \times Q_2$ . Player 1 always moves first. A fragment of the game automaton  $\mathcal{G}$  is shown in Fig. 3.

Given the game automaton  $\mathcal{G}$ , which has 370 states and 1202 transitions. The computation of player 1's winning set  $\text{Win}_1$  takes 0.05 seconds running python on a laptop with Intel Core™ 2 Duo CPU and 2GB of RAM. For a game  $(\mathcal{G}, v_0)$ , where  $v_0 \in I \cap \text{Win}_1 = \{(1, ad, \mathbf{1}, 1), (1, ce, \mathbf{1}, 1), (2, ad, \mathbf{1}, 2), (2, bf, \mathbf{1}, 2), (4, ce, \mathbf{1}, 4), (4, bf, \mathbf{1}, 4)\}$ , player 1 has a winning strategy. Hence, with full knowledge of the game, player 1 has  $\frac{|I \cap \text{Win}_1|}{|I|} = 25\%$  chance of winning. But starting just knowing that the opponent's language is a strictly 2-local language, player 1 uses the following learner:

**Definition 3 ([23]):** For all positive presentations  $\phi$ , define  $\text{GIM}_f$  as: 1)  $i = 0$ :  $\text{GIM}_f(\phi[i]) := \emptyset$ ; 2)  $\phi(i) = \#$ :  $\text{GIM}_f(\phi[i]) := \text{GIM}_f(\phi[i-1])$ ; 3) otherwise:  $\text{GIM}_f(\phi[i]) := \text{GIM}_f(\phi[i-1]) \cup f(\phi[i])$ .

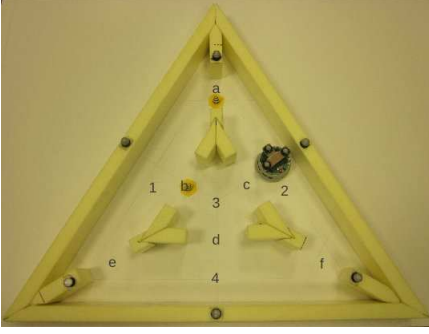
**Example 1:** Let  $f = \text{factor}_2$ . Suppose  $\text{GIM}_f(\phi[m]) = \{\#a, ab, b\#\}$  and  $\phi(m+1) = ac$ . Then  $\text{GIM}_f(\phi[m+1]) = \text{GIM}_f(\phi[m]) \cup \text{factor}_2(\#a\#) = \{\#a, ab, b\#\} \cup \{\#a, ac, c\#\} = \{\#a, ab, ac, b\#, c\#\}$ .

The learning algorithm  $\text{GIM}$  used by player 1 performs as follows: given the finite initial segment of a presentation  $\phi_2[m]$ , firstly the learner uses  $\text{GIM}_{\text{factor}_2}$  to compute a set of 2-factors  $\text{GIM}_{\text{factor}_2}(\phi_2[m])$ ; then it constructs a DFA  $\mathcal{A}_2^{[m]}$  that accepts  $L(\text{GIM}_{\text{factor}_2}(\phi_2[m]))$  with the method outlined in the Appendix.

<sup>3</sup>This assumption can be lifted, and player 1 may know nothing. She can still obtain the alphabet of player 2 through observations during the course of the game.

<sup>4</sup>For  $w = \sigma_1 \sigma_2 \dots \sigma_n \in \Sigma^*$ , the *shuffle ideal* of  $w$  is  $\Sigma^* \sigma_1 \Sigma^* \sigma_2 \dots \Sigma^* \sigma_n \Sigma^*$ .

Fig. 2: A physical implementation of the game. A Khepera II is player 1.



By removing initial and final states from  $\mathcal{A}_2^{[m]}$  we obtain  $\mathcal{A}_2^{[m]}$ , based on which the interacting function  $U_2(\cdot)$  is updated and sequentially the game  $\mathcal{G}^{[m]}$  is obtained (Fig. 1).

Figure 4 shows that the learner converges after approximately 42 games, with  $n = 125$  moves. The probability  $p$  in the learning procedure of Section IV is set to 0 which means no exploration if the current state is not initial. Table I shows the outcomes of repeated games in three different scenarios: (a) *Full-Knowledge*: player 1 knows  $A_2$  exactly; (b) *No Learning*: player 1 has no knowledge of  $A_2$  and no ability of learning; and (c) *Learning*: player 1 starts without a model of player 2 but utilizes  $\text{GIM}_{\text{factor}_2}$ . Initial conditions are chosen randomly. For the *no learning* case, player 1 does not win even once in 300 games. When she has full knowledge of the game our data indicates a winning ratio of 27%, which is close to the theoretical value of 25%. When player utilizes a learner, she reaches a win ratio of 26%.

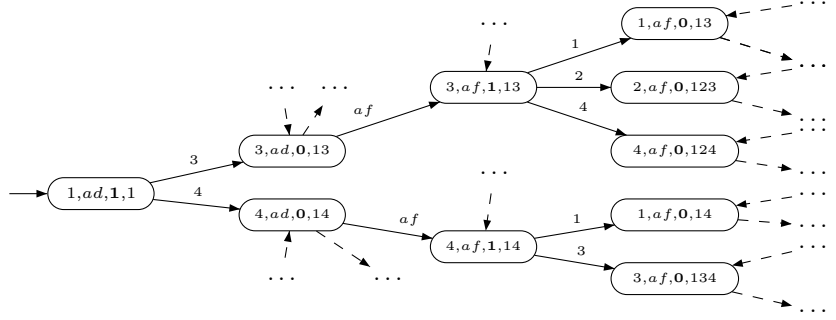
## VI. DISCUSSION AND CONCLUSION

This note demonstrates the use of grammatical inference for planning in two-player zero-sum games involving uncertain finite-state transition systems that interact adversarially. Starting with an incomplete model of the adversary, a player iteratively updates the model based on observations of the opponent's behavior, using an appropriate learning algorithm selected based on whatever prior knowledge is available. If none is available, a hypothesis about the class of models the adversary dynamics belongs to is made. If the correct hypothesis is made, and a characteristic sample of the opponent's behavior (language) is observed, the behavior of the learned model converges to the actual player 2 behavior in finitely many steps. As the adversary model becomes more accurate, strategy development is increasingly more effective. In the proposed architecture, learning and control are combined in a modular way, in the sense that a range of different methods can be adapted and used, in conjunction with grammatical inference, for control synthesis.

## APPENDIX

Given a grammar  $G$  in the form of a set of  $k$ -factors that generates a Strictly  $k$ -Local language, a finite state automaton accepting the language described with  $G$  is obtained through the following procedure: first consider a (non)-canonical finite state automaton that accepts  $\Sigma^*$ :  $\mathcal{D}_k = \langle Q_D, \Sigma, T_D, \{\lambda\}, F_D \rangle$ , where (i)  $Q_D = \text{Pr}^{\leq k-1}(\Sigma^*)$ ; (ii)  $T_D(u, a) = \text{Sf}^{=k-1}(ua)$  iff  $|ua| \geq k-1$  and  $ua$  otherwise; (iii)  $\lambda$  is the initial state, and (iv)  $F_D = Q_D$  is the set of final states (all states are final). We refer to  $\mathcal{D}_k$  as the  $\text{SL}_k$  finite state automaton for  $\Sigma^*$ . Figure 5(a) shows the  $\text{SL}_3$  finite state automaton for  $\Sigma^*$ . Given a  $\text{SL}_k$  grammar  $G$ , a (non)-canonical finite state automaton

Fig. 3: A fragment of the game automaton  $\mathcal{G} = (A_1 \circ A_2) \times \mathcal{A}_s = \langle V, \Sigma_1 \cup \Sigma_2, T, I, F \rangle$  for the door-robot game. A state, for example,  $(3, af, 1, 13)$  means the robot is in room 3, doors  $a$  and  $f$  is closed, now it is robot's turn ( $t = 1$ ) and the rooms has been visited is  $\{1, 3\}$ . The transition  $T_1(i, (i, j)) = j$  in  $A_1$  is briefly expressed by  $T_1(i, j) = j$ .



accepting  $L(G)$  can be obtained by removing some transitions and the finality of some of the states<sup>5</sup> in  $\mathcal{D}_k$  [25]. Given a  $\text{SL}_3$  grammar  $G = \{\#aa, \#ab, aab, aaa, aba, ba\#\}$ , the output of the learner is given in Fig. 5(b). For example, transition  $ba \xrightarrow{b} ab$  is removed because  $bab$  is not in  $G$ .

## REFERENCES

- [1] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas, "Symbolic planning and control of robot motion," *IEEE Robotics Automation Magazine*, vol. 14, no. 1, pp. 61–70, 2007.
- [2] 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.
- [3] 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.
- [4] N. Piterman and A. Pnueli, "Synthesis of reactive(1) designs," in *In Proceedings of Verification, Model Checking, and Abstract Interpretation*. Springer, 2006, pp. 364–380.
- [5] 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.
- [6] 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.
- [7] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive, high-level robot control," *Robotics Automation Magazine, IEEE*, vol. 18, no. 3, pp. 65–74, sept. 2011.
- [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] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.
- [10] 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.
- [11] J. de Kleer and B. C. Williams, "Diagnosing multiple faults," *Artif. Intell.*, vol. 32, no. 1, pp. 97–130, Apr. 1987.
- [12] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
- [13] 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.
- [14] X. Yang, M. Lemmon, and P. Antsaklis, "Inductive inference of optimal controllers for uncertain logical discrete event systems," in *Proceedings of the 1995 IEEE International Symposium on Intelligent Control*, aug 1995, pp. 585–590.
- [15] K. Chatterjee and T. A. Henzinger, "A survey of stochastic  $\omega$ -regular games," *Journal of Computer and System Sciences*, vol. 78, pp. 394–413, 2012.

<sup>5</sup>Removing finality of a state  $q$  in  $\mathcal{A}$  means to remove  $q$  from the set of final states in  $\mathcal{A}$ .

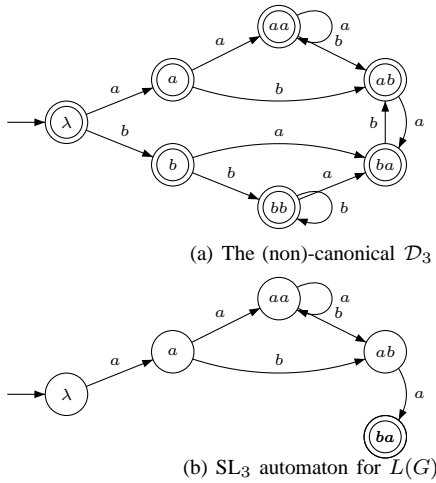


Fig. 5: The (non)-canonical automaton  $\mathcal{D}_3$  accepting  $\Sigma^*$  for  $\Sigma = \{a, b\}$  (top) and the  $SL_3$  automaton obtained for  $L(G)$ , where  $G = \{\#aa, \#ab, aab, aaa, aba, ba\#\}$ , after removing transitions and the finality of some states (bottom).

[16] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*.

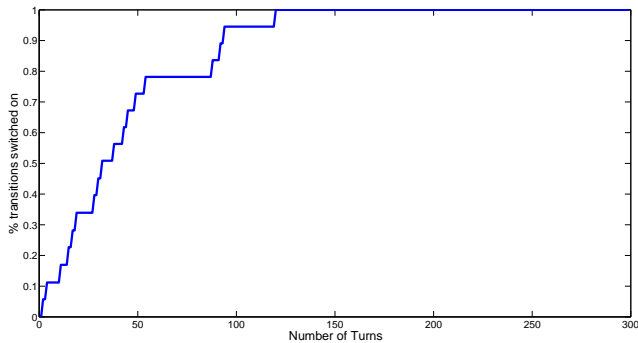


Fig. 4: Ratio of transitions learned versus all possible adversary transitions, in terms of number of turns played.

Kuwer, 1999.

- [17] W. Thomas, “Infinite games and verification (extended abstract of a tutorial),” in *Proceedings of the 14th International Conference on Computer Aided Verification*, ser. CAV ’02. London, UK, UK: Springer-Verlag, 2002, pp. 58–64.
- [18] E. M. Gold, “Language identification in the limit,” *Information and Control*, vol. 10, no. 5, pp. 447–474, 1967.
- [19] P. Garcia, E. Vidal, and J. Oncina, “Learning locally testable languages in the strict sense,” in *Proceedings of the Workshop on Algorithmic Learning Theory*, 1990, pp. 325–338.
- [20] J. Heinz, A. Kasprzik, and T. Kötzing, “Learning with lattice-structured hypothesis spaces,” *Theoretical Computer Science*, vol. 457, pp. 111–127, 2012.
- [21] D. Perrin and J. É. Pin, *Infinite words: automata, semigroups, logic and games*. Elsevier, 2004.
- [22] Y. Shoham and K. Layton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.
- [23] 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.
- [24] P. Caron, “Langage: A maple package for automaton characterization of regular languages,” *Theoretical Computer Science*, vol. 231, no. 1, pp. 5 – 15, 2000. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0304397599000134>
- [25] J. Heinz, “Inductive learning of phonotactic patterns,” Ph.D. dissertation, University of California, Los Angeles, 2007.

	Games	Wins
<i>No learning</i>	300	0
<i>Learning</i>	300	79
<i>Full knowledge</i>	300	82

TABLE I: Comparison results with three types of player 1. For the case of *no learning*, player 1 eventually moves out of her winning set.