Engineering Applications of Artificial Intelligence 37 (2015) 378–391

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Symbolic planning and control using game theory and grammatical inference

Jie Fu [c], Herbert G. Tanner [a,*], Jeffrey N. Heinz [b], Konstantinos Karydis [a], Jane Chandlee [b], Cesar Koirala [b]

[a] Mechanical Engineering, University of Delaware, Newark, DE 19716, United States
[b] Linguistics and Cognitive Science, University of Delaware, Newark, DE 19716, United States
[c] Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104, United States

ARTICLE INFO

Article history:
Received 27 December 2013
Received in revised form
22 September 2014
Accepted 29 September 2014

Keywords:
Grammatical inference
Temporal logic control
Adaptive systems

ABSTRACT

A system can accomplish an objective specified in temporal logic while interacting with an unknown, dynamic but rule-governed environment, by employing grammatical inference and adapting its plan of action on-line. The purposeful interaction of the system with its unknown environment can be described by a deterministic two-player zero-sum game. Using special new product operations, the whole game can be expressed with a factored, modular representation. This representation not only offers computational benefits but also isolates the unknown behavior of the dynamic environment in a particular subsystem, which then becomes the target of learning. As the fidelity of the identified environment model increases, the strategy synthesized based on the learned hypothesis converges in finite time to the one that satisfies the task specification.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

One of the central problems in developing autonomous systems is to ensure that they satisfy their performance specifications even when operating in *unknown, dynamic and potentially adversarial* situations. This problem is not adequately addressed currently; part of the success of industrial robotics, for instance, is that robots operate in structured, known, carefully-controlled environments. The underlying theory that supports robot deployment and guarantees performance includes stringent assumptions on the structure and nature of the robots' workspace.

This paper shows how algorithmic game theory and grammatical inference can be jointly utilized to synthesize adaptive control policies for agents operating in unknown, dynamic, and potentially adversarial environments with respect to temporal logic constraints. In this context, agents exhibit behaviors that can be captured by some abstract, purely discrete deterministic models, such as automata, Kripke structures, or transition systems (Clarke et al., 1999). To a single agent, all other agents become part of an antagonistic environment. The goal is to synthesize control policies, or strategies, that ensure that an agent satisfies a temporal logic specification without fully knowing a priori the environment which it reacts to.

This type of integration of machine learning with control design can potentially be applied to the performance analysis and supervision of several complex industrial systems. In practice, such systems are composed of multiple black-box, heterogeneous components (Van Kranenburg et al., 2008). Due to imprecise knowledge of the dynamics of the individual components as well as their specifications, it is difficult to verify the correctness of supervisory strategies. This paper proposes a paradigm for (1) constructing a model for the system using observations and some prior knowledge of its components and (2) controlling the overall system in a way that decouples control from identification. This allows for the application of legacy supervisory controllers to systems with modified or new components. Potential applications include fault diagnosis in transportation systems (Mortellec et al., 2013), the design of air-traffic control systems (Whittle et al., 2005), manufacturing (Feng et al., 2007), and software verification (Ivancic et al., 2011).

From a technical standpoint, much of the work in planning when the environment dynamics is uncertain can be found in the context of reinforcement learning (RL) (Sutton and Barto, 1998; Werbos, 1991; Bertsekas and Tsitsiklis, 1996; Lewis et al., 2012), in which the entire system is modeled as a Markov decision process (MDP). When machine learning is applied for multi-agent coordination or strategy development in cooperative or noncooperative games in the presence of uncertainty (Brafman and Tennenholtz,

* Corresponding author.
E-mail addresses: jief@seas.upenn.edu (J. Fu), btanner@udel.edu (H.G. Tanner), heinz@udel.edu (J.N. Heinz), kkaryd@udel.edu (K. Karydis), janemc@udel.edu (J. Chandlee), koirala@udel.edu (C. Koirala).

http://dx.doi.org/10.1016/j.engappai.2014.09.020
0952-1976/© 2014 Elsevier Ltd. All rights reserved.

Nomenclature

$\mathsf{GIM} : \mathsf{SEQ} \to \mathsf{REP}$	a learning algorithm that takes the first i elements of a presentation and returns a grammar.
$(\mathcal{G}, v_0)$	an initialized two-player turn-based game with the initial state v_0.
#	a pause.
ℓ	a literal, which is either an atomic proposition $\alpha \in \mathcal{AP}$ or the negation of a proposition $\neg\alpha$.
$\Gamma : Q \to 2^{\Sigma}$	an active event function, mapping a state to a set of actions enabled at that state.
Γ_A	an active event function of semiautomaton A.
λ	the empty string.
$\mathbb{B}$	a Boolean variable denoting whose turn it is to play: $\mathbb{B} = \mathbf{1}$ for player 1, $\mathbb{B} = \mathbf{0}$ for player 2.
$\mathcal{AP}$	a set of atomic logical propositions.
$\mathcal{C}$	the set of world states, defined to be the set of all conjunctions of literals.
$\mathcal{G}$	a two-player turn-based game.
Alg	a learning algorithm that identifies a game $\mathcal{G}$ in the limit from positive presentation.
Attr(X)	the attractor for the set X.
LB	a labeling function that maps a state $q \in Q$ into a subset of $\mathcal{AP}$.
Pr(L)	the prefixes of a language L.
REP	the class of possible grammars.
SA(GIM)	the set of SAs identifiable in the limit from positive presentations by the normal-form learner GIM.
SEQ	the set of all finitely long initial portions of all possible presentations of all possible L.
S_i	a memoryless, deterministic strategy for player i in the game $\mathcal{G}$.
WS_i	a winning strategy for player i.
Ω	the task specification, given as a logical formula over $\mathcal{AP}$.
$\phi : \mathbb{N} \to L \cup \{\#\}$	a positive presentation of L.
$\phi[i]$	the first $i+1$ elements of ϕ.
ρ	a run, which is a finite (or infinite) state sequence.
Σ	a finite alphabet.
Σ^*	a set of finite sequences with alphabet Σ.
Σ^{ω}	a set of infinite sequences with alphabet Σ.
Σ^n	a set of sequences with alphabet Σ of length n.
$\Sigma^{\leq n}$	a set of sequences with alphabet Σ of length less than n.
Post(σ)	the post-condition of action σ, which is a conjunction of literals.
Pre(σ)	the pre-condition of action σ, which is a conjunction of literals.
Acc	the acceptance component of an automaton.
$\mathsf{Val}_v(\mathsf{S}_1, \mathsf{S}_2)$	the value of game from v under strategies $\mathsf{S}_1, \mathsf{S}_2$ for players 1 and 2.
Win_i	the winning region of player i.
$A_i = \langle Q_i, \Sigma_i, T_i, \mathcal{AP}, \mathsf{LB}_i \rangle$*normal*	an LTS captures the dynamics of the agent ($i=1$) or its environment ($i=2$).
L	the language, which is a subset of Σ^*.
$L(\mathcal{A})$	the language of $\mathcal{A}$.
$L(\mathcal{G})$	the language of game $\mathcal{G}$, which is a set of strings that generates all possible finite runs in $\mathcal{G}$.
$L(\mathcal{G}, v_0)$	the language of an initialized game $(\mathcal{G}, v_0)$.
$L(G)$	the language generated by grammar G.
$L_i(\mathcal{G})$	the language of player $i \in \{1,2\}$ in game $\mathcal{G}$.
$L_i(\mathcal{G}, v_0)$	the language of player $i \in \{1,2\}$ in the initialized game $(\mathcal{G}, v_0)$.
$P = A_1 \circ A_2$	the turn-based product with A_1 and A_2.
Q	a finite set of states.
$T(q_1, \sigma)!$	a transition with label σ from q_1 is defined.
$T : Q \times \Sigma \to Q$	a transition function.
U_i	the interaction function, which maps a pair of states (q_i, q_j) to the set of actions which player j can no longer initiate at state q_j.
$w(i)$	the $i+1^{th}$ symbol in a word w.
$w \in \Sigma^{\omega}$	an ω-word.
Inf(w)	the set of symbols occurring infinitely often in w.
last(w)	the last symbol of w.
Occ(w)	the set of symbols occurring in w.
$\mathbb{N}$	the set of natural numbers.
$\lvert w \rvert$	the length of a string w.

2003; Kash et al., 2011; Duan et al., 2007; Fang et al., 2014; Wang and de Silva, 2008), in the vast majority of cases the method of choice is some variant of RL.

In our formulation, uncertainty is rooted in lack of knowledge, not in chance. The underlying agent models are not MDPs, but deterministic transition systems. When environment interaction is uncontrollable and not probabilistic, a pure two-player game arises. Such a game cannot be reduced to an MDPs—control synthesis for an MDPs corresponds to solving a one-player stochastic game. The two classes of games are fundamentally different (Chatterjee and Henzinger, 2012).

We approach the learning problem through grammatical inference (GI), instead of RL. Grammatical inference is of particular interest to problems involving control synthesis with temporal logic because it naturally applies to formal objects such as automata, formal languages, and discrete transition systems and provides a class of algorithms that identify them. We use a particular learning paradigm within GI: *identification in the limit from positive data* (Gold, 1967). This choice is motivated by the fact that in our problem formulation the agent observes the unknown process without being able to query or being otherwise informed about what *cannot* happen. Grammatical inference differs from RL in at least two main aspects: in RL, the sets of states and transitions of the entire system are known, and the dynamics of the environment are stochastic; in our problem setting, the environment is modeled as a *deterministic* transition system with unknown states and transition relation. Another conceptual difference is that RL addresses the question of *what actions* maximize a reward related to the given objective, while GI aims at *identifying* the unknown context in which the system operates.

Although it might be possible, it is not clear how RL methods could be applied to the scenarios studied here. One challenge is how to specify the reward functions with respect to tasks specified in terms of temporal logic. There is some limited work in this direction (Thiébaux et al., 2006), but it remains a open question. Grammatical inference conveniently decouples learning from planning and control, because it does not dictate the strategy but merely identifies the unknown components of the dynamics. Whatever the control method of choice, as GI progressively increases the fidelity of the model, the effectiveness of the control strategy is bound to increase too.

The paper is organized as follows. Section 2 provides a general, high-level overview of the technical formulation and elaborates on the technical challenges that its development has presented. Section 3 introduces the relevant notation and terminology and lays the mathematical foundation for the subsequent technical

discussion. Section 4 presents a factor-based algorithm for control synthesis with respect to temporal logic constraints. In Section 5, grammatical inference is integrated with control synthesis to construct a reactive, adaptive controller in an unknown environment. Here we prove the correctness of the strategy computation and the convergence of the learner. We demonstrate this convergence using an example case study in Section 6. Section 7 concludes the paper by reviewing the approach in a more general context and discusses alternative formulations and future extensions.

2. Overview

The paper deals with the following technical problem.

Problem 1. Consider an agent A_1 with a task specification φ, interacting with an unknown, dynamic, and hostile environment A_2. How can A_1 come to know the nature of A_2 and use this knowledge to determine whether a plan of action for A_1 exists that ensures φ is satisfied?

Part of the challenge in the proposed approach to this problem is the integration of theoretical elements from different fields—particularly, temporal logic control and grammatical inference—into a cohesive methodology. Our approach is to combine the various elements of the overall system in a modular way. Let us follow Fig. 1: at the high level, we abstract the concrete system and its dynamical environment into finite-state transition systems. Then, in order to combine the system, its task specification, and its unknown dynamic environment, into a two-player game, we defined novel product operations.

Since the true environment is unknown, the agent hypothesizes a model for it and the game being played. Then, based on the game hypothesis, the agent synthesizes a strategy (i.e., controller) to satisfy its specification (i.e., the winning condition). The environment model may be crude and naive in the beginning, but as the agent collects more observations, its grammatical inference module refines the environment model. Under certain conditions, correct prior knowledge for the model and a characteristic set for the observation data and the model structure, the fidelity of the continuously updated model increases to a point where the agent can compute sure-winning strategies to achieve the goal, whenever these exist.

The advantages of this approach are fourfold. First, we can exploit the structure of the individual system components so that the representation of the total game is polynomially smaller in the number of factors. Second, we show that a winning strategy can be computed directly from this factored representation of the game. Third, we can isolate the unknown behavior of the dynamic environment into a single model, which is the target of the learning algorithm. Fourth, different types of GI algorithms can be applied to learn the behavior of the environment under different conditions without imposing constraints on the method to be used for control: learning is decoupled from planning.

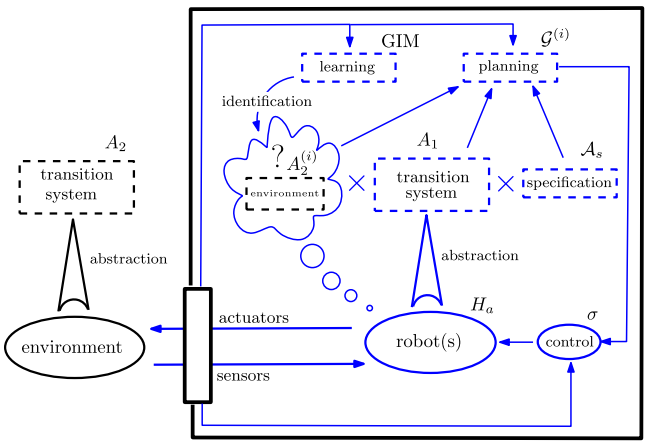

Fig. 1. The architecture of hybrid planning and control with a module for grammatical inference. The products that combine a model of the system, a model of a dynamical environment and a temporal logic control objective are defined in text.

3. Mathematical preliminaries for formal languages

In this section, we introduce some technical background on formal languages and automata theory (Hopcroft et al., 2006). Readers familiar with this material may skip this section on a first reading and refer to it as needed.

Let Σ denote a fixed, finite alphabet, and Σ^n, $\Sigma^{\leq n}$, Σ^*, Σ^ω be sequences over this alphabet of length n, of length less than or equal to n, of any finite length, and of infinite length, respectively. The *empty string* is denoted λ, and the *length of string* w is denoted $|w|$. A *language* L is a subset of Σ^*. The prefixes of a language L are denoted $\mathrm{Pr}(L)=\{u\in\Sigma^*|(\exists w\in L)(\exists v\in\Sigma^*)[uv=w]\}$. A word $w\in\Sigma^\omega$ is called an *ω-word*. Given an ω-word w, $\mathrm{Occ}(w)$ denotes the set of symbols occurring in w, and $\mathrm{Inf}(w)$ is the set of symbols occurring infinitely often in w. Given a finite word $w\in\Sigma^*$, $\mathrm{last}(w)$ denotes the last symbol of w. We refer to the $i+1^{th}$ symbol in a word w by writing $w(i)$; the first symbol in w is indexed with $i=0$.

An semiautomaton (SA) deterministic in transitions is a tuple $A=\langle Q,\Sigma,T\rangle$ where Q is a finite set of states, Σ is a finite alphabet, and $T:Q\times\Sigma\to Q$ is the transition function. The transition $T(q_1,\sigma)=q_2$ is also written as $q_1\xrightarrow{\sigma}q_2$, and is expanded recursively in the usual way. We write $T(q_1,\sigma)!$ to express that $T(q_1,\sigma)$ is defined. An *active event function* $\Gamma:Q\to 2^\Sigma$ is defined as $\Gamma(q):=\{\sigma\in\Sigma|T(q,\sigma)!\}$. We denote with $\Gamma_A(\cdot)$ the active event function of SA A. 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) where $\rho(i)\in Q$ for each i and $\rho(i+1)=T(\rho(i),w(i))$, $i\geq 0$. A run of A on word w is denoted ρ_w. An SA A can be *completed* by adding a state sink such that for all $q\in Q$, and for all $\sigma\in\Sigma\setminus\Gamma_A(q)$, $T(q,\sigma)=\mathrm{sink}$. For all $\sigma\in\Sigma$, let $T(\mathrm{sink},\sigma)=\mathrm{sink}$. This operation ensures that $\forall(q,\sigma)\in Q\times\Sigma, T(q,\sigma)!$.

Consider now a quintuple $\mathcal{A}=\langle Q,\Sigma,T,I,\mathrm{Acc}\rangle$ where $\langle Q,\Sigma,T\rangle$ is an SA deterministic in transitions, I is the set of *initial states*, and Acc is the *acceptance component*. Different types of Accs give rise to Grädel et al. (2002):

- finite state automaton (FSA), in which case $\mathrm{Acc}=F\subseteq Q$, and $\mathcal{A}$ accepts $w\in\Sigma^*$ if the run $\rho_w\in Q^*$ satisfies $\rho_w(0)\in I$ and $\mathrm{last}(\rho_w)\in F$, and
- Büchi automata, in which case $\mathrm{Acc}=F\subseteq Q$, and $\mathcal{A}$ accepts $w\in\Sigma^\omega$ if the run $\rho_w\in Q^\omega$ satisfies $\rho_w(0)\in I$ and $\mathrm{Inf}(\rho_w)\cap F\neq\emptyset$.

The set of (in)finite words accepted by $\mathcal{A}$ is the *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$. An automaton is *deterministic* if it is deterministic in transitions and I is a singleton. In this case, with a slight abuse of notation, we denote with I the single initial state. A deterministic finite-state automaton with the smallest number of states recognizing a language L is called a *canonical* automaton (acceptor) for L. Unless otherwise specified, we understand that A is the SA *obtained* from a FSA $\mathcal{A}$ by unmarking the initial state and final states in $\mathcal{A}$.

Let $\mathcal{AP}$ be a set of atomic logical propositions. A labeled finite-state transition system, also known as a *Kripke structure* (Clarke et al., 1999), is a tuple $TS=\langle Q,\Sigma,T,\mathcal{AP},\mathrm{LB}\rangle$ where $\langle Q,\Sigma,T\rangle$ is a SA, and $\mathrm{LB}:Q\to 2^{\mathcal{AP}}$ is a labeling function that maps a state $q\in Q$ into a subset of $\mathcal{AP}$. A Kripke structure can be obtained as an abstraction of some concrete dynamical system, through a variety

of different abstraction methods (Tiwari, 2008; Clarke et al., 2003; Alur et al., 2003; Abate et al., 2011).

Given a set of atomic propositions, we use Linear Temporal Logic (LTL) formulae (Emerson, 1990) to specify a set of desired system properties such as safety, reachability, and liveness. In particular, we consider a fragment of LTL; a formula in this fragment can be equivalently expressed as a language over $2^{\mathcal{AP}}$, accepted by a deterministic FSA (DFA) or a deterministic Büchi automaton. In cases when the specification is represented by a DFA, we have a reachability objective; for a DBA, it is Büchi.

4. Strategizing in a game using its factors

In this section, we see how the winning strategies (i.e., controllers) of a player with reachability or Büchi objectives can be computed, if they exist. Though the solution to the strategy synthesis problem for two-player zero-sum games with these objectives is known (Grädel et al., 2002), the algorithms computing these strategies take a complete representation of the game as input. This poses problems when not every aspect of the game is known in advance. Therefore, Section 5 will revisit this problem after relaxing the requirement of full knowledge of the game.

4.1. Two-player deterministic games

First, we briefly review the definition of deterministic, turn-based, two-player zero-sum games with perfect information and the solution of such games. For more details, the reader is referred to Grädel et al. (2002).

Definition 1 (*Grädel et al., 2002*). A two-player turn-based zero-sum game is a tuple $\mathcal{G}=\langle V_1 \cup V_2, \Sigma_1 \cup \Sigma_2, T, I, F\rangle$, where (1) V_i is the set of states where player i moves, (2) Σ_i is the set of actions for player i, $V_1 \cap V_2 = \Sigma_1 \cap \Sigma_2 = \varnothing$, $V = V_1 \cup V_2$; (3) $T : V_i \times \Sigma_i \to V_j$ is the transition function where $(i,j) \in \{(1,2),(2,1)\}$; (4) I is the set of initial game states, and (5) $F \subseteq V_1 \cup V_2$ is the *winning condition*: in reachability (resp. safety or Büchi) games: a run ρ is winning for player 1 if $\mathrm{Occ}(\rho) \cap F \neq \varnothing$ (resp. $\mathrm{Occ}(\rho) \subseteq F$ for safety, and $\mathrm{Inf}(\rho) \cap F \neq \varnothing$ for Büchi).

An *initialized game*, denoted $(\mathcal{G}, v_0)$, is the game $\mathcal{G}$ with a designated initial state $v_0 \in I$. A *memoryless*, *deterministic strategy* for player i in game $\mathcal{G}$ is a function $\mathsf{S}_i : V_i \to \Sigma_i$ which for a given state $v \in V_i$, outputs an action $\sigma \in \Sigma_i$ enabled from v for player i to take. Player i *follows* strategy S_i if it plays the action $\mathsf{S}_i(v)$ at state v.

For reachability and Büchi games, a memoryless winning strategy always exists for one of the players (Grädel et al., 2002). Thus, in this paper, when a strategy is mentioned, we mean memoryless, deterministic strategy. A strategy WS_i is *winning* for player i, if and only if for every finite prefix $\rho \in V^* V_i$ in $(\mathcal{G}, v_0)$, if player i follows WS_i, then player i wins the game, obtains a payoff 1, and its opponent obtains a payoff -1. The *winning region* of player i, denoted Win_i, is the set of states from which she has a winning strategy.

Reachability and Büchi games are determined, and only one player has a pure winning strategy. For these games, strategies are deterministic because the initial configuration of the game determines exactly one of two players has a winning strategy (Grädel et al., 2002; Perrin and Éric Pin, 2004). Given the game starting at the state $v \in V$, policy S_i for player i, $i=1,2$, the *value* of the game is $\mathsf{Val}_v(\mathsf{S}_1, \mathsf{S}_2) \in \{(1,-1),(0,0),(-1,1)\}$, where (u_1, u_2) is the payoff vector, u_1 for player 1 and u_2 for player 2. For any state v, if neither player plays his winning strategy even if it is in his winning region, then $\mathsf{Val}_v(\mathsf{S}_1, \mathsf{S}_2) = (0,0)$. If v is in the winning region of player 1, for any strategy S_2 for player 2, $\mathsf{Val}_v(\mathsf{WS}_1, \mathsf{S}_2) = (1,-1)$. If v is in the winning region of player 2, then for any strategy S_1 for player 1, $\mathsf{Val}_v(\mathsf{S}_1, \mathsf{WS}_2) = (-1,1)$. By the worst case assumption on the behavior of player 2, player 2 follows WS_2 whenever WS_2 is defined from the current state. If v is in the winning region of player 1 yet player 1 follows $\mathsf{S}_1 \neq \mathsf{WS}_1$, then it will run into a state in Win_2 for which WS_2 is defined and the value of the game under $\mathsf{S}_1, \mathsf{WS}_2$ is again $(-1,1)$.

For a game $\mathcal{G}=\langle V_1 \cup V_2, \Sigma_1 \cup \Sigma_2, T, I, F\rangle$ and for a set of states $X \subseteq V$, the *attractor* (Grädel et al., 2002) of X, denoted $\mathrm{Attr}(X)$, is the largest set of states $W \supseteq X$ in $\mathcal{G}$ from which player 1 can force a run into X. It is defined recursively as follows. Let $W_0 = X$ and set

$$W_{i+1} := W_i \cup \{v \in V_1 | (\exists \sigma \in \Gamma_{\mathcal{G}}(v))[T(v,\sigma) \in W_i]\} \cup \{v \in V_2 | (\forall \sigma \in \Gamma_{\mathcal{G}}(q))[T(v,\sigma) \in W_i]\}, \quad \text{for } i \geq 0, \ i \in \mathbb{N}. \tag{1}$$

Let $\mathsf{WS}_1(v) = \sigma$ for the state $v \in V_1$ and action $\sigma \in \Gamma_{\mathcal{G}}(v)$ identified above.

Since $\mathcal{G}$ is finite, there exists a smallest $m \in \mathbb{N}$ such that $W_{m+1} = W_m = \mathrm{Attr}(X)$ and the strategy WS_1 obtained above ensures player 1 can reach a state in X in finitely many steps.

If $\mathcal{G}$ is a reachability game, the winning region of player 1 is $\mathsf{Win}_1 = \mathrm{Attr}(F)$ and the winning region of player 2 is $\mathsf{Win}_2 = V \setminus \mathsf{Win}_1$. Player 1 has a memoryless winning strategy if the game starts at some initial state $v_0 \in \mathsf{Win}_1 \cap I$. In the case where $\mathcal{G}$ is a Büchi game, the winning region of player 1, Win_1, is the attractor of a recurrent set, which is the set of states player 1 can force the game to visit infinitely often (Grädel et al., 2002). For both reachability and Büchi games, the attractor is central to determining the winning region of player 1. The time complexity of solving reachability and Büchi games are $\mathcal{O}(m+n)$ and $\mathcal{O}(n(m+n))$, respectively, where m is the total number of transitions and $n = \|V\|$ in $\mathcal{G}$.

4.2. Constructing the game

Let $\mathcal{AP}$ be the set of atomic propositions describing *world states* (or the state of the combined agent-environment system). Given $\mathcal{AP}$, a *literal* ℓ is defined to be either an atomic proposition $\alpha \in \mathcal{AP}$ or the negation of a proposition $\neg\alpha$. The set of world states $\mathcal{C}$ is defined to be the set of all conjunctions of literals, i.e. $\mathcal{C} = \{c = \ell_1 \wedge \ell_2 \ldots \wedge \ell_n | (\exists \alpha \in \mathcal{AP})(\forall i \in [1,\ldots,n]) \quad [\ell_i = \alpha \vee \ell_i = \neg\alpha]\}$, such that, for any $c \in \mathcal{C}$, each proposition in $\mathcal{AP}$ appears at most once.

Assume now that the behavior of both the agent (player 1) and its environment (player 2) can be captured by some labeled transition system (LTS), $A_1 = \langle Q_1, \Sigma_1, T_1, \mathcal{AP}, \mathsf{LB}_1\rangle$ for player 1, and $A_2 = \langle Q_2, \Sigma_2, T_2, \mathcal{AP}, \mathsf{LB}_2\rangle$, for player 2, where for $i=1,2$, each component $\langle Q_i, \Sigma_i, T_i\rangle$ is a SA, and $\mathsf{LB}_i : Q_i \to \mathcal{C}$ is a labeling function. We assume the actions of both players have conditional effects over $\mathcal{AP}$, which means that an action $\sigma \in \Sigma_i$ can be taken if a certain pre-condition $\mathrm{Pre}(\sigma) \in \mathcal{C}$ over $\mathcal{AP}$ is satisfied; then after the action is concluded, a certain post-condition $\mathrm{Post}(\sigma) \in \mathcal{C}$ is known to be satisfied.

The labeled transition system A_i resembles a Kripke structure over $\mathcal{AP}$, only here the definition of labeling function is slightly different. For any $\sigma \in \Gamma_{A_i}(q)$, $\mathsf{LB}_i(q) \Longrightarrow \mathrm{Pre}(\sigma)$, and for any $\sigma \in \Sigma_i$ and $q \in Q_i$, if there exists $q' \in Q_i$ such that $T_i(q', \sigma) = q$, then we have $\mathsf{LB}_i(q) \Longrightarrow \mathrm{Post}(\sigma)$.

Without loss of generality, we assume that the alphabets of A_1 and A_2 are disjoint, i.e. $\Sigma_1 \cap \Sigma_2 = \varnothing$. Player i can give up his turn, in which case she "plays" a generic (silent) symbol $\epsilon \in \Sigma_i$. In the specific games discussed in this paper, player 1 is not allowed to give up turns (since it would be disadvantageous to do so), but player 2 (the adversary) can. Therefore, we have $\epsilon \notin \Sigma_1$ and $\epsilon \in \Sigma_2$.

To capture how each player can interfere with the other, we define the *interaction functions* $U_i : Q_i \times Q_j \to 2^{\Sigma_j}$, for $(i,j) \in \{(1,2),(2,1)\}$ as

$$U_i(q_i, q_j) = \{a \in \Gamma_{A_j}(q_j) | \mathsf{LB}_i(q_i) \wedge \mathsf{LB}_j(q_j) \Longrightarrow \neg \mathrm{Pre}(a)\}.$$

An interaction function U_i maps a pair of states (q_i, q_j) to the set of actions which player j can no longer initiate at state q_j.

Definition 2 (*Turn-based product*). Given a LTS for each player, $A_1 = \langle Q_1, \Sigma_1, T_1, \mathcal{AP}, \mathsf{LB}_1\rangle$ and $A_2 = \langle Q_2, \Sigma_2, T_2, \mathcal{AP}, \mathsf{LB}_1\rangle$, and their interacting functions U_1, U_2, the turn-based product $P = \langle Q_p, \Sigma_1 \cup \Sigma_2, T_p, \mathcal{AP}, \mathsf{LB}_p\rangle$ is an LTS denoted $A_1 \circ A_2$, defined as follows:

$Q_p = Q_1 \times Q_2 \times \{\mathbf{0}, \mathbf{1}\}$ is the set of states, where the last component is a Boolean variable $\mathbb{B} \in \{\mathbf{0}, \mathbf{1}\}$ denoting whose turn it is to play: $\mathbb{B} = \mathbf{1}$ for player 1, $\mathbb{B} = \mathbf{0}$ for player 2.

T_p is the transition relation. $T_p((q_1, q_2, \mathbb{B}), \sigma) = (q_1', q_2, \mathbf{0})$ if $\mathbb{B} = \mathbf{1}$, $q_1' = T_1(q_1, \sigma)$, with $\sigma \notin U_2(q_2, q_1)$ and $T_p((q_1, q_2, \mathbb{B}), \sigma) = (q_1, q_2', \mathbf{1})$ if $\mathbb{B} = \mathbf{0}$, $q_2' = T_2(q_2, \sigma)$, with $\sigma \notin U_1(q_1, q_2)$.

$\mathsf{LB}_p : Q_p \to \mathcal{C}$ is the labeling function, where for $(q_1, q_2, \mathbb{B})$, $\mathsf{LB}_p(q_1, q_2, \mathbb{B}) = \mathsf{LB}_1(q_1) \wedge \mathsf{LB}_2(q_2)$.

The time complexity of constructing P is $\mathcal{O}(\|Q_1 \times Q_2\| \times k)$, where $k = \max(\|\Sigma_1\|, \|\Sigma_2\|)$.

If one includes a silent action ϵ in Σ_i for $i = 1,2$, the players may not necessarily play in turns—as in the specific case of agent-environment interaction considered here. We let $\textsc{Pre}(\epsilon) = \textsc{Post}(\epsilon) = \texttt{True}$, that is, we assume that an ϵ action cannot change the world state.

The task specification is given as a logical formula Ω over $\mathcal{AP}$. Through the labeling function LB_p, Ω can take the form of a language over Q_p, accepted by a *completed deterministic* automaton $\mathcal{A}_s = \langle S, Q_p, T_s, I_s, F_s\rangle$ where $\mathsf{sink} \in S$. Intuitively, the task specification encoded in $\mathcal{A}_s$ specifies a set of histories over the world states. Here subscript s distinguishes $\mathcal{A}_s$ as the automaton for the system's *specification*—the specification automaton.

The turn-based product P gives snapshots of different stages in a game. It does not capture any of the game history that led to this stage. We overcome the lack of memory in P by using another product operation.

Definition 3 (*Two-player turn-based game automaton*). Given the turn-based product $P = \langle Q_p, \Sigma_1 \cup \Sigma_2, T_p, \mathcal{AP}, \mathsf{LB}_p\rangle$ and the specification automaton $\mathcal{A}_s = \langle S, Q_p, T_s, I_s, F_s\rangle$ that encodes formula φ, the two-player turn-based game automaton is a (special) product of P and $\mathcal{A}_s$, denoted $\mathcal{G} = P \ltimes \mathcal{A}_s = (A_1 \circ A_2) \ltimes \mathcal{A}_s = \langle V_1 \cup V_2, \Sigma_1 \cup \Sigma_2, T, I, F\rangle$, such that

$V_1 \subseteq Q_1 \times Q_2 \times \{\mathbf{1}\} \times S$ is the set of states where player 1 makes a move and $V_2 \subseteq Q_1 \times Q_2 \times \{\mathbf{0}\} \times S$ is the set of states where player 2 makes a move. We write $V \stackrel{\text{def}}{=} V_1 \cup V_2$.

Σ_i is the set of actions of player $i \in \{1, 2\}$. We write $\Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$.

T is the transition relation defined as follows:

$$(\forall (q_1, q_2, \mathbb{B}, s) \in V)[(\forall \sigma \in \Gamma_P((q_1, q_2, \mathbb{B})))$$
$$[T_p((q_1, q_2, \mathbb{B}), \sigma) = (q_1', q_2', \mathbb{B}') \ \wedge \ T_s(s, (q_1', q_2', \mathbb{B}')) = s'$$
$$\Longrightarrow T((q_1, q_2, \mathbb{B}, s), \sigma) = (q_1', q_2', \mathbb{B}', s')]].$$

The semantics of the transition relation is defined as follows: given a state $(q_1, q_2, \mathbb{B}, s)$, if there exists an action σ enabled at $(q_1, q_2, \mathbb{B})$ in the turn-based product, and $T_p((q_1, q_2, \mathbb{B}), \sigma) = (q_1', q_2', \mathbb{B}')$, then by taking the action σ, the game arrives at a state $(q_1', q_2', \mathbb{B}', s')$. State $s' = T_s(s, (q_1', q_2', \mathbb{B}))$ keeps track of the progress made in this transition with respect to the objective expressed in $\mathcal{A}_s$ and the previous state s in $\mathcal{A}_s$.

$I = \{(q_1, q_2, \mathbf{1}, s) \in V | s = T_s(I_s, (q_1, q_2, \mathbf{1}))\}$ is the set of possible initial game states.

$F = \{(q_1, q_2, \mathbb{B}, s) \in V | s \in F_s\}$ is the winning condition. If $\mathcal{A}_s$ is a FSA and game $\mathcal{G}$ is a reachability game, then a run $\rho \in V^*$ is winning for player 1 iff $\mathsf{last}(\rho) \in F$; if $\mathcal{A}_s$ is a DBA, and the game is a Büchi game, then a run $\rho \in V^\omega$ is winning for player 1 iff $\mathsf{Inf}(\rho) \cap F \neq \varnothing$.

The reachability (resp. Büchi) objective for the system is expressed by formula φ. We say player 1 achieves its reachability (resp. Büchi) objective if there exists a winning strategy for player 1 in the corresponding reachability (resp. Büchi) game. With P and $\mathcal{A}_s$ the game automaton $\mathcal{G}$ is constructed in time $\mathcal{O}(\|Q_p \times S\| \times m)$ where m is the number of transitions in P.

4.3. Synthesis using the game's factors

Given $\mathcal{G}$, the winning strategy of player 1, if it exists, can be computed with the methods of Section 4.1. Noting that $\mathcal{G}$ is built from separate components (factors) $\mathcal{G} = (A_1 \circ A_2) \ltimes \mathcal{A}_s$, a natural question to ask is whether there exists a method for computing the winning strategy WS_1 without obtaining the whole game automaton $\mathcal{G}$: can WS_1 be computed using the factors $A_1, A_2, \mathcal{A}_s$ of $\mathcal{G}$? The answer is yes. The attractor can be computed for any $X \subseteq Q_1 \times Q_2 \times \{\mathbf{1}, \mathbf{0}\} \times S$ using A_1, A_2, and $\mathcal{A}_s$ by recasting its definition in terms of these factors, the interacting functions, and the product operations.

Theorem 1. *Consider a game $\mathcal{G} = (A_1 \circ A_2) \ltimes \mathcal{A}_s$ with states $V \subseteq Q_1 \times Q_2 \times \{\mathbf{1}, \mathbf{0}\} \times S$ and some $X \subseteq V$. Let $Y_0 = X$ and*

$$Y_{n+1} = Y_n \cup \{(q_1, q_2, \mathbf{1}, s) | (\exists \sigma \in \Sigma_1)[T_1(q_1, \sigma) = q_1' \wedge T_s(s, (q_1', q_2, \mathbf{0})) = s' \wedge \sigma \notin U_2(q_2, q_1) \wedge (q_1', q_2, \mathbf{0}, s') \in Y_n \ ]\}$$
$$\cup \{(q_1, q_2, \mathbf{0}, s) | (\forall \sigma \in \Sigma_2)[T_1(q_2, \sigma) = q_2' \wedge T_s(s, (q_1, q_2', \mathbf{1})) = s' \wedge \sigma \notin U_1(q_1, q_2) \wedge (q_1, q_2', \mathbf{1}, s') \in Y_n]\}. \quad (2)$$

Let $W_0, W_1, \ldots$, be the sequence of sets of states obtained from (1) on page 9 with $W_0 = X$. Then for all $n \in \mathbb{N}$, $Y_n = W_n$.

Proof. The proof is by induction on n. First given $X \subseteq Q_1 \times Q_2 \times \{\mathbf{1}, \mathbf{0}\} \times S$, $X \cap V \subseteq V$ is the set of reachable states in $\mathcal{G}$. $W_0 = X \cap V = Y_0 \cap V$. Next we assume $W_n = Y_n \cap V$ and show $W_{n+1} = Y_{n+1} \cap V$. Consider any $v = (q_1, q_2, \mathbb{B}, s) \in Y_{n+1} \cap V$:

Case 1. $v \in Y_n \cap V$. Then $v \in W_n$ and therefore belongs to W_{n+1}.

Case 2. $v \notin Y_n \cap V$ and $\mathbb{B} = \mathbf{1}$. Then $\exists \sigma \in \Sigma_1$, $\sigma \notin U_2(q_2, q_1)$ such that $T_1(q_1, \sigma) = q_1'$ and $T_s(s, (q_1', q_2, \mathbf{0})) = s'$ and $v' = (q_1', q_2, \mathbf{0}, s') \in Y_n$. Moreover, $v \in V$ implies $v' \in V$ as v' is reachable from v. Hence $v' \in Y_n \cap V$. By definition of the products $\circ$ and $\ltimes$, it follows that $\sigma \in \Gamma_{\mathcal{G}}$ and $T(v, \sigma) = v' \in W_n$. Therefore $v \in W_{n+1}$.

Case 3. $v \notin Y_n \cap V$ and $\mathbb{B} = \mathbf{0}$. The argument here is similar to case 2.

The cases are exhaustive and in each case $v \in W_{n+1}$. Hence $Y_{n+1} \subseteq W_{n+1}$. The argument that $W_{n+1} \subseteq Y_{n+1}$ follows similarly. □

Corollary 1. *There is an $m \in \mathbb{N}$ such that the fixed point $Y_{m+1} = Y_m$ coincides with* $\mathsf{Attr}(X)$.

Theorem 1 and its corollary show that the computation of the attractor $\mathsf{Attr}(X)$ for a given set of states X can be achieved using the individual factors of the game. Thus it is not necessary to compute the game automaton $\mathcal{G}$, an advantage since $\mathcal{G}$ can be significantly larger in size than any of its factors.

Below is a procedure which implements the factor-based method to compute the attractor. Given $X \subseteq Q_1 \times Q_2 \times \{\mathbf{1}, \mathbf{0}\} \times S$, let $Y_0 = X$, and Y_{n+1} is computed from Y_n in two-steps:

1. let $\mathsf{Pre}(Y_n) = \bigcup_{v \in Y_n} \mathsf{Pre}(v)$ where $\mathsf{Pre}((q_1, q_2, \mathbb{B}, s)) =$

$$\begin{cases} \mathbb{B} = \mathbf{1}: \\ \{(q_1, q_2', \mathbf{0}, s') | T_s(s', (q_1, q_2, \mathbf{1})) = s \wedge (\exists \sigma \in \Sigma_2 \setminus U_1(q_1, q_2'))[T_2(q_2', \sigma) = q_2]\}, \\ \mathbb{B} = \mathbf{0}: \\ \{(q_1', q_2, \mathbf{1}, s') | T_s(s', (q_1, q_2, \mathbf{0})) = s \wedge (\exists \sigma \in \Sigma_1 \setminus U_2(q_2, q_1'))[T_1(q_1', \sigma) = q_1]\}. \end{cases} \quad (3)$$

In other words, $\mathsf{Pre}(Y_n)$ includes a set of states of $\mathcal{G}$, from each of which there exists *at least one* outgoing transition that leads to a state in Y_n.

2. $Y_{n+1} = Y_n \cup \{\mathsf{Pre}(Y_n) \setminus \{(q_1, q_2, \mathbf{0}, s) \in \mathsf{Pre}(Y_n) | (\exists \sigma \in \Sigma_2 \setminus U_1(q_1, q_2))$
$[T_2(q_2, \sigma) = q_2' \wedge T_s(s, (q_1, q_2', \mathbf{1})) = s' \wedge (q_1, q_2', \mathbf{1}, s') \notin Y_n]\}\}$.

That is, given state $v \in \mathsf{Pre}(Y_n)$ where player 2 makes a move, if there exists at least one transition from v that leads to a state outside Y_n, then v is not included in Y_{n+1}. The fixpoint is $Y_m = Y_{m+1} = Y$ and $Y \cap V = \mathsf{Attr}(X \cap V)$.

Since the winning region of a Büchi game is in fact the attractor of a recurrent target state (see Grädel et al., 2002), for both reachability and Büchi games the winning region of player 1, Win_1, can be computed using the factors. This method exploits the definitions of the two products and the factors of $\mathcal{G}$ to directly compute the attractor for a given set of states. WS_1 is the same as if $\mathcal{G}$ had been computed. The reduction in computational complexity when computing the winning region is due to the fact that a state in $Q_1 \times Q_2 \times \{\mathbf{0}, \mathbf{1}\} \times S$ is searched only when it is in the winning region. In the worst case, the winning region Win_1 includes the whole state set of $\mathcal{G}$, and in this case the factor-based method offers no computational benefit.

5. Identifying the game

Player 1 can accomplish her task if and only if (1) she has full knowledge of the dynamics of player 2, and (2) the game starts at the initial state in $\mathsf{Win}_1 \cap I$. The objective of this section is to show how player 1 can learn the true nature of the game she is playing, if player 2 is rule-governed. We first introduce the notion of *identification in the limit of semiautomata* and show that, provided the true model of player 2 is identifiable in the limit in this way, player 1 can learn the game in the limit and then plan her actions effectively.

5.1. Identification in the limit from positive presentations

We start with some background on the concept of identification in the limit from positive presentations. Informally, a positive presentation of a language L is a sequence of words belonging to the language, interspersed with pauses (i.e., moments in time when no information is forthcoming). Formally, a *positive presentation* of L is a total function $\phi : \mathbb{N} \to L \cup \{\#\}$, where # denotes a pause, such that for every $w \in L$, there exists $n \in \mathbb{N}$ such that $\phi(n) = w$ (Jain et al., 1999). A presentation ϕ can be understood as an infinite sequence $\phi(0)\phi(1)\cdots$ that contains every element of L. Let $\phi[i] \equiv \{\phi(k)\}_{k=0}^{i} \equiv \phi(0)\phi(1)\ldots\phi(i)$ denote the first $i+1$ elements of ϕ, and let SEQ denote the set of all finitely long initial portions of all possible presentations of all possible L.

Grammars are finite representations of potentially infinite languages. We denote with REP the class of possible grammars. Let $L(G)$ be the language generated by grammar G. A learner (otherwise referred to as a learning algorithm, GIM) takes the first i elements of a presentation and returns a grammar: $\mathsf{GIM} : \mathsf{SEQ} \to \mathsf{REP}$. The grammar returned by GIM represents the learner's *hypothesis* of the language.

A learner GIM *identifies in the limit from positive presentations* a class of languages $\mathcal{L}$ if and only if for all $L \in \mathcal{L}$, for all presentations ϕ of L, there exists a $n \in \mathbb{N}$ such that for all $m \geq n$, $\mathsf{GIM}(\phi[m]) = G$ and $L(G) = L$ (Gold, 1967). When GIM converges on ϕ this way, we write $\mathsf{GIM}(\phi) = G$. The learner does not necessarily return the target grammar, but rather one that generates the same language as the target (i.e., the two grammars are *language-equivalent*.) For distinct presentations of L, the language-equivalent grammars returned by GIM may *also* be distinct.

To preclude this latter possibility, in this paper we consider *normal-form* learners. A GIM which identifies $\mathcal{L}$ in the limit from positive presentations is *a normal-form learner* if for all languages $L \in \mathcal{L}$ and for all distinct presentations ϕ, ϕ' of L, $\mathsf{GIM}(\phi) = \mathsf{GIM}(\phi')$. Note that any learner for a class of regular languages can be converted into a normal-form learner by transforming its output grammars into canonical automata.

A characterization of the language classes that are identifiable in the limit from positive data is available in Angluin (1980). See also Jain et al. (1999) and de la Higuera (2010) for additional details about this learning paradigm and a comparison to alternatives.

5.2. What game am I playing?

We have seen that synthesizing a strategy has a solution if player 1 has full knowledge of the game being played. Suppose, however, that player 1 has knowledge of her own capabilities and objective, but does not have full knowledge of the capabilities of player 2. How can player 1 plan effectively given her uncertainty about the game she is playing? Here we let player 1 make inferences about the game *over time* based on the actions of player 2. While player 1 may not make the best moves at first, given enough observations, she will eventually determine which game is being played.

In order to define learning *of* games, it is necessary to be clear about the kind of data presentations learners must succeed on. This means being equally clear about the "language of a game." Intuitively, the data available to game-learners should be initial, finite sequences of game-play. Therefore we define the language of a game $\mathcal{G}$ to be $L(\mathcal{G}) = \{w \in \Sigma^* | T(I, w)\downarrow\}$, that is, a set of strings that generates all possible finite runs in $\mathcal{G}$.

The language of an initialized game $(\mathcal{G}, v_0)$ is defined as $L(\mathcal{G}, v_0) = \{w \in \Sigma^* | T(v_0, w)\downarrow\}$. Note that prefixes of ω-languages of deterministic Büchi automata form a regular languages (Perrin and Éric Pin, 2004); therefore, for the kinds of winning conditions and games considered in this paper, this definition always defines a regular language. Observe further that languages of games are always prefix-closed. Under this definition, a learner cannot distinguish Büchi games from reachability games since both describe regular languages. With respect to the language of a game $L(\mathcal{G})$, the projection of $L(\mathcal{G})$ on Σ_i, denoted $L_i(\mathcal{G})$, is the language of player $i \in \{1, 2\}$ in game $\mathcal{G}$.

The following definition makes explicit the idea of learning two–player deterministic games over time.

Definition 4. An algorithm Alg *identifies* $\mathcal{G}$ *in the limit from positive presentations* if and only if for all positive presentations ϕ of $L(\mathcal{G})$, there is a $m \in \mathbb{N}$ such that for all $n \geq m$, $\mathsf{Alg}(\phi[n]) = \mathcal{G}$. Algorithm Alg identifies a class of games GAMES in the limit from positive presentations if and only if, for every $\mathcal{G} \in \mathsf{GAMES}$, Alg identifies $\mathcal{G}$ in the limit from positive presentations.

The similarities between Definition 4 and identification in the limit from positive presentations of languages (Gold, 1967) should be clear. However, there is an important difference. Definition 4 requires the learning algorithm to return the target game, not just a language-equivalent one, which motivates us to seek solutions in the form of a normal-form learner. This is driven in part by the fact that our definition of the language of a game above does not distinguish between Büchi and reachability games.

The question now is whether there are algorithms that can identify classes of games in the limit from positive presentations of their plays. The next section shows how this problem reduces to the problem of identification of languages in the limit from positive data under certain conditions. Thus, for every class of formal languages identifiable in the limit from positive data, there are corresponding classes of games which are also identifiable.

5.3. Incorporating grammatical inference

The whole class of regular languages is not identifiable in the limit from positive data (Gold, 1967), and so player 1 may eventually have to employ a grammatical inference module GIM which learns some subclass of the regular languages. This will be of little solace if the correct model of player 2 is outside this subclass. Preferably, the choice of learning algorithm should be made based on some kind of prior knowledge of player 2.[1] We are not concerned with the case when the model to be learned falls outside of the class of models learnable by a GIM; it would be unreasonable to ask an algorithm to perform on a case outside its domain of validity.[2] Rather, we focus on demonstrating that when the true model is learnable through a GIM, strategies can be computed and eventually applied effectively as if the model were known in advance.

Definition 5. Let $\mathcal{L}$ be a class of languages identifiable in the limit from positive presentation by a *normal-form* learner GIM, the output of which is an FSA. Then we say that an SA $A=\langle Q,\Sigma,T\rangle$, where $\mathsf{sink}\notin Q$, is identifiable in the limit from positive presentations if for *any* $q_0\in Q$, the language accepted by FSA $\mathcal{A}=\langle Q,\Sigma,T,q_0,Q\rangle$ is in $\mathcal{L}$, and given a positive presentation ϕ of $L(\mathcal{A})$, there exists an $m\in\mathbb{N}$ such that $\forall n\geq m$, $\mathsf{GIM}(\phi[m])=\mathsf{GIM}(\phi[n])=\mathcal{A}$. The learner $\mathsf{GIM}_{\mathsf{SA}}$ for SA A is constructed from the output of GIM by unmarking the initial and final states.

Let SA(GIM) be the set of SAs identifiable in the limit from positive presentations by the normal-form learner GIM.

Now given a SA A_1, a specification automaton $\mathcal{A}_s$, and a class of semiautomata SAs, define the class of games

$$\mathsf{GAMES}(A_1,\mathcal{A}_s,\mathsf{SAs})=\{\mathcal{G}|(\exists A_2\in\mathsf{SA})[\mathcal{G}=(A_1\circ A_2)\ltimes\mathcal{A}_s]\}.$$

For this class of games we have the following result.

Theorem 2. *If, for all* $A_2\in\mathsf{SA}(\mathsf{GIM})$*, there exists* $\mathcal{A}_2\in\mathsf{range}(\mathsf{GIM})$ *such that* $L(\mathcal{A}_2)=L_2((A_1\circ A_2)\ltimes\mathcal{A}_s)$*, then* $\mathsf{GAMES}(A_1,\mathcal{A}_s,\mathsf{SA}(\mathsf{GIM}))$ *is identifiable in the limit from positive presentations.*

Proof. For any game $\mathcal{G}\in\mathsf{GAMES}(A_1,\mathcal{A}_s,\mathsf{SA}(\mathsf{GIM}))$ and any data presentation ϕ of $L(\mathcal{G})$, denote with $\phi_2(n)$ for $n\in\mathbb{N}$ the projection of $\phi(n)$ on Σ_2. Then define a learning algorithm Alg as follows:

$$\forall\phi,\forall n\in\mathbb{N},\quad \mathsf{Alg}(\phi[n])=(A_1\circ\mathsf{GIM}_{\mathsf{SA}}(\phi_2[n]))\ltimes\mathcal{A}_s.$$

We show that Alg identifies $\mathsf{GAMES}(A_1,\mathcal{A}_s,\mathsf{SA}(\mathsf{GIM}))$ in the limit.

Consider any game $\mathcal{G}\in\mathsf{GAMES}(A_1,\mathcal{A}_s,\mathsf{SA}(\mathsf{GIM}))$. Since $\mathcal{G}$ is in GAMES, there is an $A_2\in\mathsf{SA}(\mathsf{GIM})$ such that $\mathcal{G}=(A_1\circ A_2)\ltimes\mathcal{A}_s$. Consider now any data presentation ϕ of $L(\mathcal{G})$. Then ϕ_2 is a data presentation of $L_2(\mathcal{G})$. By assumption, there exists $\mathcal{A}_2\in\mathsf{range}(\mathsf{GIM})$ such that $L(\mathcal{A}_2)=L_2(\mathcal{G})$. Thus ϕ_2 is also a data presentation of $L(\mathcal{A}_2)$. Therefore, there is $m\in\mathbb{N}$ such that for all $n\geq m$, $\mathsf{GIM}_{\mathsf{SA}}(\phi_2[n]))=A_2$. Consequently, there is $m'=2m$ such that for all $n\geq m'$, $\mathsf{Alg}(\phi[n])=(A_1\circ A_2)\ltimes\mathcal{A}_s=\mathcal{G}$.

Since $\mathcal{G}$ and ϕ are selected arbitrarily, the proof is completed. □

Once the learning algorithm identifies the game, we can ensure that the controller synthesized based on the hypothesis of player 1 is correct and wins the game, no matter the strategy of her opponent. The convergence of the control law can occur prior to that of the learning algorithm; control strategies can still be effective even if computed based on slightly inaccurate game hypotheses.

5.4. Online adaptive synthesis

This section shows how strategies can adapt automatically during play in repeated games. A proof is given that through this process the strategy of player 1 eventually converges to a winning one, whenever the latter exists. For this to happen, we need to assume that the game can be played indefinitely, and that player 2 keeps playing his best even when he finds himself in a losing position.

While updating the hypothesis for the opponent dynamics by observing game plays, the agent updates her strategy as shown in Algorithm 1 on page 22. Let us walk through the steps of the algorithm, making references to the corresponding line numbers in Algorithm 1. We start the game at a random initial state $v_0\in I$, and player 1 computes a winning strategy in its hypothesized game (line 3). When the current state is within the hypothesized winning region of player 1, the player takes the action indicated by her winning strategy (line 9). Otherwise, there are two possibilities for player 1: (i) the current state is actually in the true winning region, but the incomplete knowledge of the opponent model misclassified it, or (ii) the current state is indeed out of the true winning region. In any case, player 1 can consider exploring the game with a random action with some probability p (line 14), in which case she can observe the response of player 2 and update her opponent's model (line 17). Alternatively, and with probability $1-p$, she can resign. Then the game is restarted from a random initial state (line 12).

If the repeated game is played in this way, we can show that player 1 eventually learns the true nature of the game and is therefore capable of winning the game when possible.

[1] In practice, another option is to equip player 1 with multiple learning algorithms which are each geared toward a different subclass of regular languages. Player 1 can then try to leverage the multiple hypotheses in some fashion.

[2] de la Higuera (2010) quotes Charles Babbage at the beginning of chapter 19: On two occasions I have been asked [by members of Parliament], 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

Algorithm 1. `LearnToWin`.

Input: The initial state v_0, the agent A_1, a task specification $\mathcal{A}_s$, and an upper limit N on the number of total turns (including both players' turns) to be taken.

Output: The winning region Win_1 and winning strategy WS_1 in the agent's hypothesis; and the number of games won by player 1 in the repeated game play.

begin
  $\phi(0) \leftarrow \epsilon$, $i \leftarrow 0$, $v \leftarrow v_0$, $count \leftarrow 0$, $wins \leftarrow 0$;
  $\mathcal{G}^{(i)} \leftarrow Alg(\phi[i])$, $(\mathsf{Win}_1^{(i)}, \mathsf{WS}_1^{(i)}) \leftarrow$ `ControlSynthesis` $(\mathcal{G}^{(i)})$;
  while $count \leq N$ **do**
    $count \leftarrow count + 1$;
    if $v \in V_1$ `/* The agent's turn */`
    then
      if $v \in \mathsf{Win}_1^{(i)}$ **then**
        $a \leftarrow \mathsf{WS}_1^{(i)}(v)$;
      else
        if `Sample` $([0,1]) > p$ **then**
          $v \leftarrow$`Sample` (I);
        else
          $a \leftarrow$ `Sample` $\{\sigma \in \Sigma_1 \mid \sigma$ is enabled on $v\}$;
    else
      $a \leftarrow \mathsf{S}_2(v)$; `/* The environment takes an action. */`
      $\phi[i+1] \leftarrow$ `UpdatePresentation`$(\phi[i], a)$, $i \leftarrow i+1$,
      $\mathcal{G}^{(i)} \leftarrow \mathsf{Alg}(\phi[i])$, $(\mathsf{Win}_1^{(i)}, \mathsf{WS}_1^{(i)}) \leftarrow$ `ControlSynthesis` $(\mathcal{G}^{(i)})$;
    $v \leftarrow$ `NextState` $(\mathcal{G}, v, a)$; **if** $v \in F$ **then**
      $v \leftarrow$`Sample` (I), $wins \leftarrow wins + 1$; `/* The agent wins, and the game is restarted. */`
  return $\mathsf{Win}_1^{(N)}, \mathsf{WS}_1^{(N)}$.

For an initialized game $(\mathcal{G}, v_0)$, let $L_2(\mathcal{G}, v_0)$ be the language of player 2 in $(\mathcal{G}, v_0)$. Then a grammatical inference module, when presented with the actions of player 2 in repeated gameplay, will eventually converge to a language that captures that player's behavior:

Proposition 1. *For game $(\mathcal{G}, v_0)$, and for any $x \in L_2(\mathcal{G}, v_0)$, there exists $k \in \mathbb{N}$ such that $\forall m \geq k$, $x \in L_2(\mathcal{G}^{(m)}, v_0)$.*

Proof. Consider an arbitrary $x \in L_2(\mathcal{G}, v_0)$. There can be two cases:

Case $v_0 \notin \mathsf{Win}_1^{(i)}$ By definition of $L_2(\mathcal{G}, v_0)$, there exists an interleaving action sequence $w = \sigma_0\sigma_1 \ldots \sigma_n \in L(\mathcal{G}, v_0)$ such that the projection of w onto Σ_2 is x. Player 1 makes a move at random with probability p, and the chance that player 1 selects σ_0 is $\gamma \geq p/m$, where m is the maximal number of all possible actions for player 1 or 2 for all game states, i.e., $m = \max_{v \in V} \| \{\sigma \in \Sigma | T(v, \sigma)\downarrow\} \|$. Conditioned on player 1 playing σ_0, player 2 plays σ_1 with probability greater or equal to $1/m$. Then inductively, the probability of the finite action sequence being w is $\eta \geq \min(\gamma, 1/m)^{\| w \|}$. Furthermore, since there is no upper bound on the number of games to be repeated, the probability of *never* playing w is

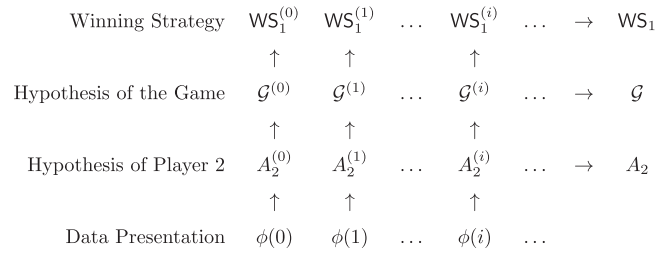

Fig. 2. Learning and planning with a grammatical inference module.

$\lim_{N\to\infty}(1-\eta)^N = 0$ where n is the number of games played. Thus eventually w will be played and the learning algorithm will update the hypothesis to $\mathcal{G}^{(k)}$ for some $k > i$. Since $w \in L(\mathcal{G}^{(k)}, v_0)$, it will be $x \in L_2(\mathcal{G}^{(k)}, v_0)$.

Case $v_0 \in \mathsf{Win}_1^{(i)}$ whatever player 2 plays, if $x \in L_2(\mathcal{G}, v_0)$, then x will be observed by player 1 and the hypothesis is updated. Otherwise, if $v_0 \notin \mathsf{Win}_1$, then player 2 will play to win, and player 1 will realize that $v_0 \notin \mathsf{Win}_1^{(k)}$ at some $k \geq i$. Since the game is repeated, in the next round the first case applies. □

Basically, the exploratory moves made by player 1 ensure that eventually a positive presentation of $L_2(\mathcal{G}, v_0)$ is obtained.

Fig. 2 illustrates how the identification in the limit of games evolves, following the identification of the formal language associated with the dynamics of player 2. This is essentially what goes on at the high level of the architecture shown in Fig. 1, page 5. Through interactions with player 2, player 1 observes the discrete evolution $\phi_2(i)$ of the dynamics of player 2. She uses the GIM to construct and update a hypothesized model $A_2^{(i)}$, together with the labeling function LB_2, and subsequently updates the interacting functions U_1 and U_2. 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 for the game being played, and uses this model to devise a winning strategy $\mathsf{WS}_1^{(i)}$. As the model of player 2 A_2 converges asymptotically to the true one, the winning strategy becomes increasingly more effective. In the limit, player 1 is guaranteed to learn the game she plays and win when the game's initial state is in Win_1.

6. Case study

6.1. The game and its class

We analyze a specific case which is representative of a class of deterministic two-player turn-based games with perfect information. This class of games is chosen because it is one of the simplest, yet non-trivial classes still capable of illustrating the main features of our analysis. Within this class there are games in which only one player has a winning strategy given the initial state.

The game being played is a repeated game; that is, the same *stage* game is being played again and again, restarting each time one of the players wins or resigns. The stage game takes place in the triangular[3] "apartment" configuration of Fig. 3. In this game the purpose of player 1, which in our simulation and experimental implementations is realized as a small mobile robot, is to visit all four rooms. The four rooms are connected with six doors that are controlled by player 2. Player 2 can close two doors at any given time, according to fixed rules that determine which door pairs are allowed to be closed and how the transition from one pair to the next is to take place. Player 2 is played either by a computer or a human in simulations, and by a human in the actual experiments.

[3] We also considered other games with four rooms arranged grid-like so that each room has two connecting doors and games where player 2 was subject to different kinds of rules.

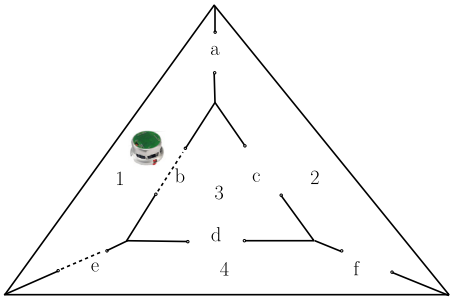

Fig. 3. The triangle room game representation.

Table 1
Some possible constraints on player 2: at each round, player 2 either does nothing or opens exactly one door and closes exactly one other so that the pair of closed doors belongs to one of the sets above.

Rules	Description
`Opposite`	Doors opposite to each other can be closed at any time: $\{a,d\},\{a,e\},\{a,f\},\{b,f\},\{c,e\},\{e,f\}$
`Adjacent`	Doors adjacent to each other can be closed at any time: $\{a,b\},\{a,c\},\{b,c\},\{b,d\},\{b,e\},\{c,d\},\{c,f\},\{d,e\},\{d,f\}$
`General`	Any pair of doors can be closed at any time.

When it is his turn, player 2 may open and close doors so that again exactly two doors are closed, or to keep the currently closed doors closed for another turn. Table 1 on page 26 shows three possible rule regimes for player 2 (others are possible).

In general, these constraints can be specified as any set of door pairs that can be closed at the same time.

Depending on the constraints on player 2, several different games can be played. An additional parameter of the game is its initial configuration. The games begin with the robot in a room with exactly two doors closed; these must be among those player 2 can close. We assume player 1 cannot choose her initial location, and that player 2 cannot choose which of the allowable pairs of doors is closed. In our simulations, the starting configuration is determined randomly.

6.2. Factoring the game

Let the set of atomic propositions describing the world states be

$$\mathcal{AP} = \{\alpha_i : \text{ robot in room } i \in \{1,2,3,4\}\} \cup \{d_{ij} : \text{ the door connecting rooms } i \text{ and } j \text{ is open}, (i,j) \in \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}\}.$$

The robot starts the game under the naive assumption that all open doors will remain open. The LTS for player 1 is $A_1 = \langle Q_1, \Sigma_1, T_1, \mathcal{AP}, \mathsf{LB}_1\rangle$ where $Q_1 = \{1,2,3,4\} = \Sigma_1$, where each element is associated with a controller that steers the robot to the corresponding room; the transition function is defined as $T_1(i,j) = j$ for $(i,j) \in \{(1,2),(1,3),(1,4),(2,1),(2,3),(2,4),(3,1),(3,2),(3,4),(4,1),(4,2),(4,3)\}$; the labeling function is same for all $q \in Q$, $\mathsf{LB}_1(i) = \alpha_i \wedge (\wedge_{(j,k)\in\{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}} d_{jk})$. We abuse notation and denote the SA extracted from LTS A_i with the same symbol; the corresponding SA is the same as the LTS it came from, less $\mathcal{AP}$ and LB_1.

Fig. 4 (left) gives a graphical representation of A_1 after the state and transition relabeling, representing that with all doors open, the robot can move from any room to any other room by initiating the appropriate motion controller.

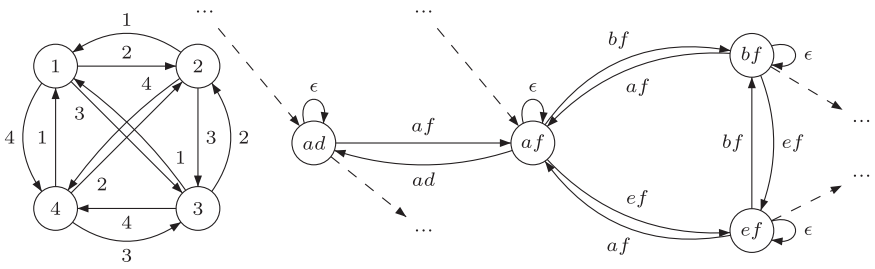

Fig. 4. SA for player 1 (left) and a fragment of the SA for player 2 (right). In A_1, the states are the rooms and the transitions are labeled with the room that player 1 is to enter. For A_2, the states are the pairs of doors that are currently closed and a transition xy indicates that doors x and y are to be closed.

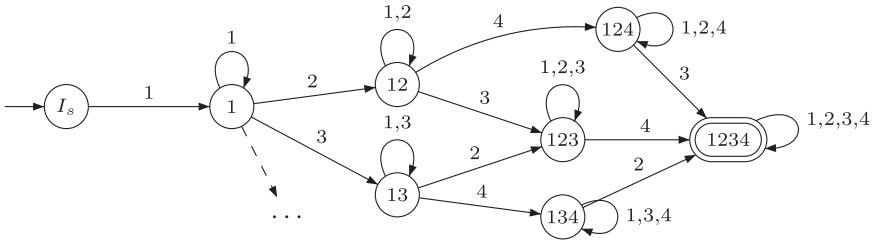

Fig. 5. A fragment of $\mathcal{A}_s$; for clarity, a transition labeled q_1 from s_1 to s_2 represents a set of transitions from s_1 to s_2 with labels $\{q_1\} \times Q_2 \times \{\mathbf{0}, \mathbf{1}\}$.

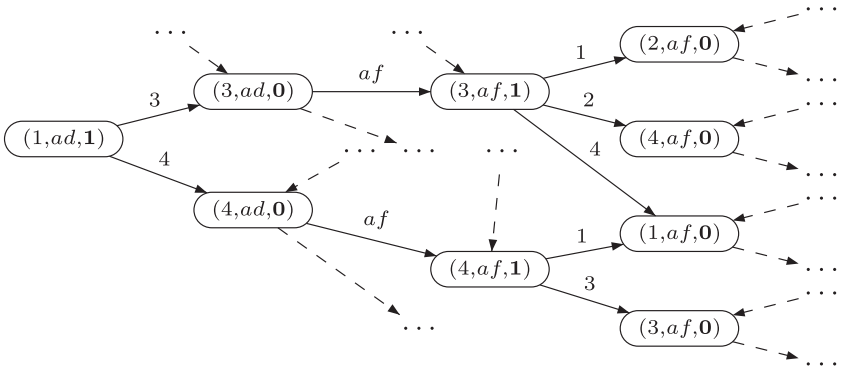

Fig. 6. A fragment of the turn-based product $P = A_1 \circ A_2 = \langle Q_p, \Sigma_1 \cup \Sigma_2, T_p, \mathcal{AP}, \mathsf{LB}_p \rangle$. State $(r, d_1 d_2, \mathbb{B})$ means player 1 is in room r, doors $\{d_1, d_2\}$ are closed, and the Boolean variable $\mathbb{B}$ indicates whose turn it is.

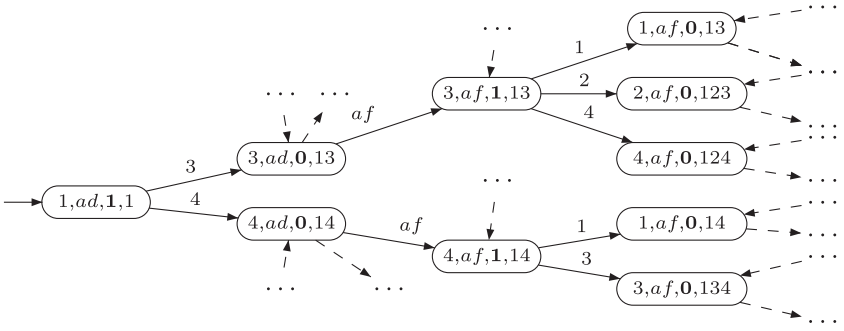

Fig. 7. A fragment of the game automaton $\mathcal{G} = (A_1 \circ A_2) \ltimes \mathcal{A}_s = \langle V, \Sigma_1 \cup \Sigma_2, T, I, F \rangle$, where $I = \{(q_1, q_2, \mathbf{1}, s) \in V | q_1 \in I_1, q_2 \in I_2, s = T_s(I_s, (q_1, q_2, \mathbf{1})) \in \{1,2,3,4\}\}$ and $F = \{(q_1, q_2, \mathbb{B}, s) \in V | s = 1234\}$.

Suppose that player 2 adheres to the `Opposite` rule in Table 1. Fig. 4 (right) then shows a fragment of the SA A_2 that models player 2.

The goal of player 1 (to visit all four rooms in any order) is described as the language accepted by $\mathcal{A}_s = \langle S, Q_p, T_s, I_s, F_s \rangle$, where $F_s = \{1234\}$. The automaton for this objective can be obtained using the minimal DFA that recognizes the union of the *shuffle ideals*[4] of the permutations of the string 1234. Each transition labeled q_1 from s_1 to s_2 can be used to generate a set of transitions from s_1 to s_2 with labels $\{q_1\} \times Q_2 \times \{\mathbf{0}, \mathbf{1}\}$. Fig. 5 shows a fragment of $\mathcal{A}_s$.

[4] For $w = \sigma_1 \sigma_2 \cdots \sigma_n \in \Sigma^*$, the *shuffle ideal* of w is $\Sigma^* \sigma_1 \Sigma^* \sigma_2 \cdots \Sigma^* \sigma_n \Sigma^*$.

The interaction functions follow from obvious physical constraints: when player 2 closes a door, player 1 cannot move through it. The interaction function $U_2(d_1d_2, r)$ gives the set of rooms player 1 cannot access from room r because doors d_1 and d_2 are closed. In Fig. 3, for instance, $U_2(ab, 1) = \{2, 3\}$. On the other hand, the actions of player 1 cannot inhibit the behavior of player 2 in any way, so $U_1(q) = \emptyset, \forall q \in Q_1 \times Q_2$. Fig. 6 shows a fragment of $A_1 \circ A_2$. A transition in $A_1 \circ A_2$, for example, $(3, ad, \mathbf{0}) \xrightarrow{af} (3, af, \mathbf{1})$, indicates that when the robot is in room 3 with doors a and d closed, if player 2 opens d and closes f, the game reaches state $(3, af, \mathbf{1})$ at which the robot has to make a move. A fragment of the game automaton $\mathcal{G}$ is shown in Fig. 7, which encodes the task specification automaton $\mathcal{A}_s$ into $A_1 \circ A_2$.

6.3. The winning region

Player 1 has a reachability objective, and therefore the winning region Win_1 is the attractor of F, denoted $\mathsf{Attr}(F)$, which is obtained by computing the fixed point of (1) on page 9 using (3) on page 14. The game automaton, a fragment of which is shown in Fig. 7, has 309 states and so is small enough to be computed directly. On the other hand, A_1, A_2, and $\mathcal{A}_s$ have 4, 6, and 16 states, respectively. Clearly the factored representation is significantly smaller than the game automaton.

The set of winning initial states for player 1 is shown below:

$$I \cap \mathsf{Attr}(F) = \{(1, ad, \mathbf{1}, 1), (1, ce, \mathbf{1}, 1), (2, ad, \mathbf{1}, 2), (2, bf, \mathbf{1}, 2), (4, ce, \mathbf{1}, 4), (4, bf, \mathbf{1}, 4)\}.$$

Interestingly, $|I \cap \mathsf{Attr}(F)|/|I|$ makes up a mere 25% of all possible initial configurations when player 2 is constrained by the `Opposite` rule in Table 1. For instance, player 1 has no winning strategy if she starts in room 3.

When different door rules are considered, cases can arise where there is no initial state from which player 1 has a winning strategy. In fact when player 2 is subject to the constraints of the `Adjacent` and `General` regimes (see Table 1), player 1 can never win, even with complete knowledge of the dynamics of player 2, because in these games $\mathsf{Attr}(F) \cap I = \emptyset$.

6.4. Employing grammatical inference

We study the case where the behavior of player 2 is characterized as a Strictly 2-Local (SL_2) language—which belongs to a subclass of the regular languages—and player 1 is equipped with an algorithm which identifies SL_2 languages in the limit from positive data. For the interested reader, in the Appendix we give a brief review of SL languages and their learners.

The grammar that a SL_2 learner outputs is the set of all contiguous subsequences of length 2 (called 2-factors) that can be found in the strings of the language. Interfacing such a learner with a strategy planner in games requires the additional step of extracting an automaton out of this grammar. Any grammar of 2-factors can be converted into a DFA which recognizes the exact same SL_2 language, which may not necessarily be the canonical acceptor for the language, but it is a *normal form* one (see Appendix).

It now follows from Theorem 2 on page 19 that if player 1 is equipped with a learning algorithm Alg that identifies SL_2 languages in the limit from positive data and outputs the normal form automaton for this class, then player 1 can identify the class of games $\mathsf{GAMES}(A_1, \mathcal{A}_s, \mathsf{SA}(\mathsf{Alg}))$ in the limit from positive data.

6.4.1. Implementing the grammatical inference module

There are several ways to implement the grammatical inference in the example considered. First, there are distinct algorithms (GIMs) for learning the SL_2 class (Garcia et al., 1990; Heinz, 2008, 2010). Second, faithfully following the description in Theorem 2 suggests that the game must be recomputed with each new hypothesis of player 2. However, performing the products with each new hypothesis is not always necessary. In some cases, as with the SL_2 case, it is possible to reduce the number of computations by precompiling a particularly useful representation of the hypothesis space for the game and performing computations upon this representation. This is precisely the manner in which we implemented the simulations. Again it serves to emphasize the point that in many cases of interest, the worst-case computational complexity can be significantly eased.

The basic idea behind our implementation of the learning algorithm Alg (described in the Appendix) follows from the observation that there is a single semiautomaton of which every semiautomaton $A_2 \in \mathsf{SA}(\mathsf{Alg})$ is a subgraph. This is the semiautomaton obtained from the normal form SL_2 acceptor for Σ^*. Player 1 has this "supergraph" semiautomaton in mind, with every transition switched "off." The moves of player 2 correspond to transitions in this supergraph. As those moves are observed, player 1 simply follows a path in the supergraph and switches specific transitions on this graph to "on." Provided a data presentation is observed, there is a point at which the graph given by the "on" transitions (and states connected to such transitions) is the target semiautomaton. We then extend this concept to the entire game automaton itself. Formally, we construct an SA that accepts $(\Sigma_2 \setminus \{\epsilon\})^*$ and keeps track of the last symbol observed —$A_2^{(0)} = \langle Q_2, \Sigma_2 \setminus \{\epsilon\}, T_2 \rangle$ where $Q_2 = \Sigma_2 \setminus \epsilon$ and $T_2(q_2, a) = a$—and define a labeling function $\mathsf{LB}_2 : Q_2 \to \mathcal{C}$ where $\mathcal{C}$ is the conjunction of literals over $\mathcal{AP}$. Informally, given $q \in Q_2$ and two doors encoded in q, d, d', then $\mathsf{LB}_2(q) = \neg[d \text{is open}] \wedge \neg[d' \text{is open}]$. Then we add self-loops $T_2(q, \epsilon) = q$, for all $q \in Q_2$. Clearly, A_2 can be obtained from $A_2^{(0)}$ by removing transitions.

We can efficiently update player 1's hypothesis of player 2's behavior and the game by introducing a switching function $\mathsf{sw} : Q_2 \times \Sigma_2 \to \{0, 1\}$. The function is initialized as $\mathsf{sw}^{(0)} : (\forall q_2 \in Q_2)[\mathsf{sw}^{(0)}(q_2, \epsilon) = 1]$ and for all σ labeling the outgoing transitions from $q_2 \in Q_2$ and $\sigma \neq \epsilon$, let $\mathsf{sw}^{(0)}(q_2, \sigma) = 0$. Basically $\mathsf{sw}^{(0)}$ encodes the initial hypothesis of player 1 that player 2 is static. Let $\mathsf{sw}^{(i)}$ denote the refinement of sw made at round i and suppose that at round $i+1$, the adversary plays σ'. This suggests $\phi(i+1) = \phi(i)\sigma'$. If $q_2 = T_2(I_2, \phi(i))$, then for all $q \in Q_2$ and $\sigma \in \Sigma_2$, $\mathsf{sw}^{(i+1)}$ can be defined by

$$\mathsf{sw}^{(i+1)}(q, \sigma) = \begin{cases} \mathsf{sw}^{(i)}(q, \sigma) & \text{if } (q, \sigma) \neq (q_2, \sigma') \\ 1 & \text{if } (q, \sigma) = (q_2, \sigma') \end{cases} \tag{4}$$

meaning that the transition from q_2 on input σ' in A_2 is now enabled. With an additional small abuse of notation, we denote with $A_2^{(i)}$ the pair $(A_2^{(0)}, \mathsf{sw}^{(i)})$, which means SA $A_2^{(0)}$ has switching function $\mathsf{sw}^{(i)}$. Graphically, $A_2^{(i)}$ is the SA obtained from $A_2^{(0)}$ by trimming the "off" transitions (where $\mathsf{sw}^{(i)}(\cdot) = 0$).

Thus, the originally hypothesized game automaton is $\mathcal{G}^{(0)} = (A_1 \circ A_2^{(0)}) \ltimes \mathcal{A}_s$. The switching function associated with $A_2^{(i)}$ can be extended naturally to $\mathcal{G}^{(i)} := \left(\mathcal{G}^{(0)}, \mathsf{sw}^{(i)}\right)$ in the following way: if $\mathsf{sw}^{(i)}(q_2, \sigma) = 1$ (resp. 0) in $A_2^{(i)}$, then $\forall v \in V_1 \cup \{v \in V_2 | v = (q_1, q_2, \mathbf{0}, s)\}$ we define $\mathsf{sw}^{(i)}(v, \sigma) = 1$ (resp. 0) in $\mathcal{G}^{(i)}$. With this extension of the switching function, the game automaton can be updated without computing *any product* during runtime. In this particular case, the pre-compilation of the game obviates the need to compute the turn-based product and game automaton every time A_2 is updated (see the complexity analysis in Section 4.2). In cases where the game automaton is too big to be computed, we expect that a similar strategy can be used with its factors.

Remark 1. In this particular example, it happens to be the case that the set of states in the game has a one to one correspondence

with the physical states of the system and its dynamic environment. However, the general framework of grammatical inference is not restricted to such cases. For example, consider a property of the environment which specifies each door cannot be closed for three consecutive turns, although the physical state of a closed door does not change, in the game arena, additional states are required for keeping track of how long a door has been kept closed. Grammatical inference will identify the set of states in the game, even though they are different from the world states, which are the conjunctions of literals.

6.5. Simulations

The initial conditions—the room where player 1 begins and which pair of doors is closed—for the game are chosen randomly. Games are played repeatedly with the most updated game model hypothesis that player 1 has constructed carrying over from game to game. The simulations below use the `Opposite` rules for player 2. A total of 300 games were played.

The convergence of the learning algorithm is measured by computing the ratio between transitions that are switched on during the repeated game versus the total number of enabled transitions in the true game automaton. Fig. 8 shows the convergence of learning using the ratio of adversary transitions that have been identified by player 1 versus the number of turns the two players have played. From this figure, we observe that after 125 turns of both players (approximately 42 games), the robot's model of the environment converges to the actual one. In fact, player 1 starts to win after the first 10 games. Moreover, after this point, player 1 wins all games that began in states in her true winning region. This result suggests that even though the model for player 2 is incomplete, the strategy synthesized by player 1 based on her hypothesis of her opponent's behavior can still be effective. Hence, the value of the game is $(-1, 1)$ for the first 10 games and $(1, -1)$ for the rest.

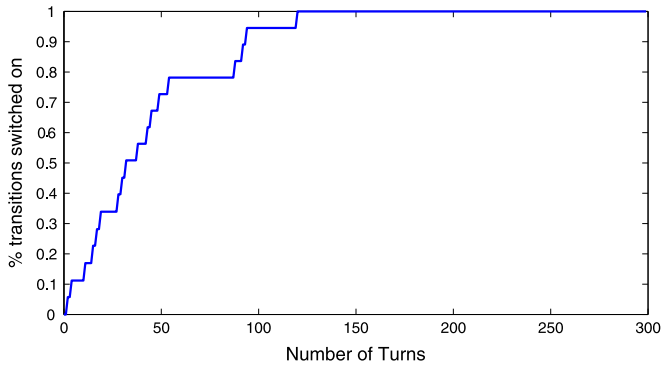

Fig. 8. The rate of convergence for the learning algorithm in a single game of 300 turns (including both players' turns.)

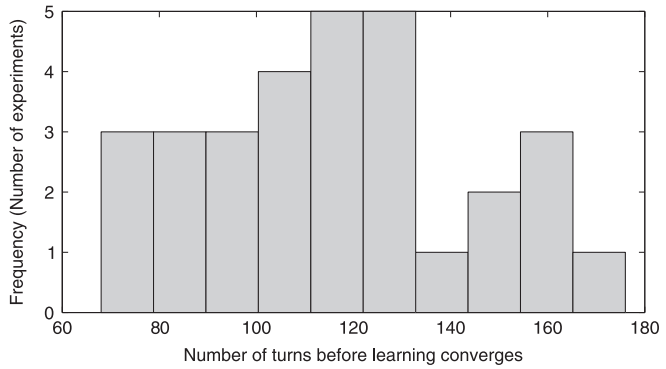

Fig. 9. Number of turns before convergence of the learning algorithm in 30 independent experiments.

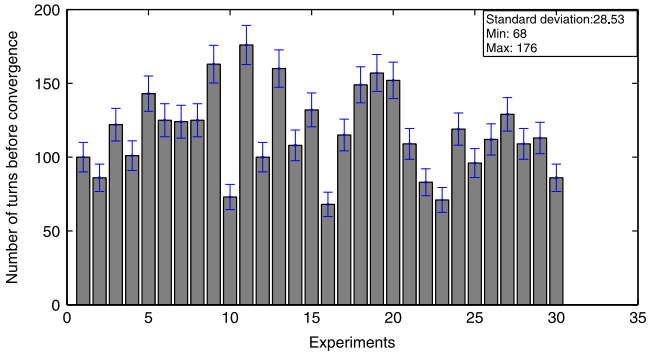

Fig. 10. Error chart of the learning algorithm with 30 independent experiments.

Figs. 9 and 10 provide a histogram and error chart showing the result of 30 independent experiments, each of which is carried out with 300 turns of two players. The standard deviation and the numbers of minimal and maximal number of turns for the learning algorithm to converge are shown in Fig. 10.

7. Concluding remarks

7.1. Conclusions and future work

This paper shows how particular classes of two-player, deterministic, zero-sum games with perfect information can be identified in the limit, leading to sure-winning strategies for the players. The prerequisites for this are (1) player 1 and her objective are known, and (2) the behavior of player 2 corresponds to a language which belongs to a class of languages identifiable in the limit from positive presentations by a normal-form learner. Provided these conditions hold, it is guaranteed that player 1 can compute a strategy which converges to the true winning strategy in the limit using positive presentations of the gameplay.

The learning results in this paper are primarily made possible by *factoring* the game according to its natural subsystems – the dynamics of player 1, the dynamics of player 2, and the objective of player 1 – which isolates the uncertainty in the game to the model of player 2. We also show that the winning strategy can be computed directly from this factorization, which can allow for a significantly more compact representation of the game, because the subsystems structure the game in a way captured by the product operations.

While this paper has focused on tools for learning subclasses of regular languages, the field of grammatical inference has made important advances in learning classes of context-free (Yokomori, 2003; Clark and Eyraud, 2007; Clark et al., 2010) and context-sensitive (Becerra-Bonache et al., 2010; Yoshinaka, 2011) languages. Players that adopt these learning algorithms will be able to identify behaviors that are not describable with finite-state descriptions and which could lead to games with infinitely many states. Infinite-state games arise in many contexts.

To extend the present research to games with imperfect information, we plan to integrate the learning methods of partial observable *action models* (the way actions affect the world) (Amir and Chang, 2008) with grammatical inference. Future work should also examine non-zero-sum games with multiple players, examine the learnability of games under different learning criteria, and explore the effectiveness of a variety of learning algorithms in specific scenarios.

7.2. Potential application to industrial systems

Many industrial control systems are assembled from (possibly third party) embedded components for which the exact dynamics are not modeled or given (Mortellec et al., 2013; Whittle et al., 2005; Feng et al., 2007; Ivancic et al., 2011). Grammatical inference can help identify the model of components as formal objects, such as, finite-state transition systems, automata, languages, we can facilitate the analysis, and ensure the correctness of control design for the overall system.

Take a particular example in the context of software verification and model checking (Ivancic et al., 2011). State of the art automated model checkers which verify that pieces of software with millions of lines of code are bug-free, usually employ a divide-and-conquer strategy: since bugs may lurk deep inside the hierarchy of processes that call each other, current computational resources of model checkers may not be sufficient to catch these bugs in a brute-force exhaustive search from the initial condition. Then, model checkers can start from a particular process deep inside the derivation tree, and explore from that point forward until they reach a pre specified *depth cutoff* limit (Ivancic et al., 2011). In such instances, the software verification tool needs to have a model of the environment that calls the process where it starts working from (referred to as the *calling environment*). It is often the case that large scale software architectures embody pieces of code from third parties, of which the input–output behavior is not known exactly. In these model checking applications, which can range from database management to aircraft control software, there is a practical need to construct and refine the calling environment model. The reported methodology may be useful in this large-scale model checking context, by treating the entry function and its calling environment as two adversaries, and contribute to the technology for calling environment model refinement.

Although in this paper we consider the formulation of a zero-sum game between a control plant and its dynamic environment, due to the independence between the learning module and the control synthesis, the grammatical inference module is not limited to adversarial interactions between components. With learning, we extract the inter-dependencies and interactions between different components, which are necessary for system integration and analysis.

Acknowledgment

This work is supported by the National Science Foundation, under award number 1035577.

Appendix

A string u is a *factor* of string w iff $\exists x, y \in \Sigma^*$ such that $w = xuy$. If in addition $|u| = k$, then u is a *k-factor* of w. The k-factor function $\text{factor}_k : \Sigma^* \to 2^{\Sigma^{\leq k}}$ maps a word w to its set of k-factors if $|w| > k$; otherwise it maps w to $\{w\}$. This function is extended to languages as $\text{factor}_k(L) := \bigcup_{w \in L} \text{factor}_k(w)$. A language L is Strictly k-Local (SL_k) if there exists a finite set $G \subseteq \text{factor}_k(\sharp\Sigma^*\sharp)$, such that $L = \{w \in \Sigma^* | \text{factor}_k(\sharp w \sharp) \subseteq G\}$, where $\sharp$ indicates the beginning and end of a string. G is the grammar that generates L.

A language is called Strictly Local if it is Strictly k-Local for some k. There are many distinct characterizations of this class. For example, it is equivalent to the languages recognized by (generalized) Myhill graphs, to the languages definable in a restricted propositional logic over a successor function, and to exactly those languages which are closed under suffix substitution (McNaughton and Papert, 1971; De Luca and Restivo, 1980; Rogers and Pullum, 2011). Furthermore, there are known methods for translating between automata-theoretic representations of Strictly Local languages and these others.

Theorem 3 (*(Garcia et al., 1990)*). *For known k, the Strictly k-Local languages are identifiable in the limit from positive data.*

Readers are referred to the cited papers for a proof of this theorem. We sketch the basic idea here with the grammars for Strictly k-Local languages defined above. Consider any $L \in \text{SL}_k$. The grammar for L is $G = \text{factor}_k(\{\sharp\} \cdot L \cdot \{\sharp\})$,[5] and G contains only finitely many strings. The learning algorithm initially hypothesizes $G = \emptyset$, and for each word $w \in L$, computes $\text{factor}_k(w)$ to add these k-factors to its current hypothesized grammar. There will be some finite point in every data presentation of L such that the learning algorithm converges to the grammar of L (because $|G|$ is finite). This particular algorithm is analyzed by Heinz (2010) and is a special case of lattice-structured learning (Heinz et al., 2012).

This learning algorithm does not output finite-state automata, but sets of factors. However, there is an easy way to convert any grammar of factors into an acceptor which recognizes the same Strictly Local language. This acceptor is not the canonical acceptor for this language, but it is a *normal form*. It is helpful to define a function $\text{suf}^k(L) = \{v \in \Sigma^k | (\exists w \in L)[(\exists u \in \Sigma^*)[w = uv]]\}$. Given k and a set of factors $G \subseteq \text{factor}_k(\{\sharp\} \cdot \Sigma^* \cdot \{\sharp\})$, construct a finite-state acceptor $\mathcal{A}_G = \langle Q, \Sigma, T, I, \text{Acc} \rangle$ as follows.

- $Q = \text{suf}^{k-1}(\text{Pr}(L(G)))$
- $(\forall u \in \Sigma^{\leq 1})(\forall \sigma \in \Sigma)(\forall v \in \Sigma^*)[T(uv, \sigma) = v\sigma \Leftrightarrow uv, v\sigma \in Q]$
- $I = \{\lambda\}$ if $L(G) \neq \emptyset$ else $\emptyset$
- $\text{Acc} = \text{suf}^{k-1}(L(G))$

The proof that $L(\mathcal{A}_G) = L(G)$ is given in Heinz (2007, p.106).

References

Abate, A., D'Innocenzo, A., Di Benedetto, M.D., 2011. Approximate abstractions of stochastic hybrid systems. IEEE Trans. Autom. Control 56 (11), 2688–2694.
Alur, R., Dang, T., Ivančić, F., 2013. Counter-example guided predicate abstraction of hybrid systems. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, pp. 208–223.
Amir, E., Chang, A., 2008. Learning partially observable deterministic action models. J. Artif. Intell. Res. 33 (1), 349–402.
Angluin, D., 1980. Inductive inference of formal languages from positive data. Inf. Comput.–Inf. Control 45, 117–135.
Becerra-Bonache, L., Case, J., Jain, S., Stephan, F., 2010. Iterative learning of simple external contextual languages. Theor. Comput. Sci. 411, 2741–2756.
Bertsekas, D.P., Tsitsiklis, J.N., 1996. Neuro-Dynamic Programming. Athena Scientific, Nashua, NH.
Brafman, R.I., Tennenholtz, M., 2003. Learning to coordinate efficiently: a model-based approach. J. Artif. Intell. Res. 19, 11–23.
Chatterjee, K., Henzinger, T.A., 2012. A survey of stochastic ω-regular games. J. Comput. Syst. Sci. 78, 394–413.
Clark, A., Eyraud, R., 2007. Polynomial identification in the limit of substitutable context-free languages. J. Mach. Learn. Res. 8, 1725–1745.
Clarke Jr., E.M., Grumberg, O., Peled, D.A., 1999. Model Checking. MIT Press, Cambridge, MA.
Clarke, E., Fehnker, A., Han, Z., Krogh, B., Ouaknine, J., Stursberg, O., Theobald, M., 2003. Abstraction and counterexample-guided refinement in model checking of hybrid systems. Int. J. Found. Comput. Sci. 14 (04), 583–604.
Clark, A., Eyraud, R., Habrard, A., 2010. Using contextual representations to efficiently learn context-free languages. J. Mach. Learn. Res. 11, 2707–2744.
de la Higuera, C., 2010. Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, New York.
De Luca, A., Restivo, A., 1980. A characterization of strictly locally testable languages and its application to subsemigroups of a free semigroup. Inf. Control 44 (3), 300–319.
Duan, Y., Liu, Q., Xu, X.H., 2007. Application of reinforcement learning in robot soccer. Eng. Appl. Artif. Intell. 20 (7), 936–950.

[5] The operator $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ concatenates string sets: given $S_1, S_2 \subseteq \Sigma^*$, $S_1 \cdot S_2 = \{xy | x \in S_1, y \in S_2\}$.

Emerson, E.A., 1990. Temporal and modal logic. Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B) 995, 1072.
Fang, M., Groen, F.C., Li, H., Zhang, J., 2014. Collaborative multi-agent reinforcement learning based on a novel coordination tree frame with dynamic partition. Eng. Appl. Artif. Intell. 27 (0), 191–198.
Feng, T., Wang, L., Zheng, W., Kanajan, S., Seshia, S., 2007. Automatic model generation for black box real-time systems. In: Design, Automation Test in Europe Conference Exhibition, 2007, pp. 1–6.
Garcia, P., Vidal, E., Oncina, J., 1990. Learning locally testable languages in the strict sense. In: Proceedings of the Workshop on Algorithmic Learning Theory, pp. 325–338.
Gold, E.M., 1967. Language identification in the limit. Inf. Control 10 (5), 447–474.
Grädel, E., Thomas, W., Wilke, T. (Eds.), 2002. Automata Logics, and Infinite Games: A Guide to Current Research. Springer-Verlag New York, Inc., New York, NY, USA.
Heinz, J., 2008. Left-to-right and right-to-left iterative languages. In: Clark, A., Coste, F., Miclet, L. (Eds.), Grammatical Inference: Algorithms and Applications, 9th International Colloquium, Lecture Notes in Computer Science, vol. 5278. Springer, Berlin, Heidelberg, p. 8497.
Heinz, J., 2007. Inductive Learning of Phonotactic Patterns (Ph.D. Thesis), University of California, Los Angeles.
Heinz, J., 2010. String extension learning. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden, pp. 897–906.
Heinz, J., Kasprzik, A., Kötzing, T., 2012. Learning with lattice-structured hypothesis spaces. Theor. Comput. Sci. 457, 111–127.
Hopcroft, J.E., Motwani, R., Ullman, J.D., 2006. Introduction to Automata Theory, Languages, and Computation, 3rd edition Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
Ivancic, F., Balakrishnan, G., Gupta, A., Sankaranarayanan, S., Maeda, N., Tokuoka, H., Imoto, T., Miyazaki, Y., 2011. DC2: a framework for scalable, scope-bounded software verification. In: IEEE/ACM International Conference on Automated Software Engineering, pp. 133–142.
Jain, S., Osherson, D., Royer, J.S., Sharma, A., 1999. Systems that Learn: An Introduction to Learning, Theory: Learning Development and Conceptual Change, 2nd edition The MIT Press, Cambridge, MA.
Kash, I.A., Friedman, E.J., Halpern, J.Y., 2011. Multiagent learning in large anonymous games. J. Artif. Intell. Res. 40, 571–598.
Lewis, F.L., Vrabie, D., Vamvoudakis, K.G., 2012. Reinforcement learning and feedback control. IEEE Control Syst. Mag., 76–105.
McNaughton, R., Papert, S., 1971. CounterFree Automata. MIT Press, Cambridge, MA.
Mortellec, A.L., Clarhaut, J., Sallez, Y., Berger, T., Trentesaux, D., 2013. Embedded holonic fault diagnosis of complex transportation systems. Eng. Appl. Artif. Intell. 26 (1), 227–240.
Perrin, D., Éric Pin, J., 2004. Infinite Words: Automata, Semigroups, Logic and Games. Elsevier, London, UK.
Rogers, J., Pullum, G., 2011. Aural pattern recognition experiments and the subregular hierarchy. J. Logic Lang. Inf. 20, 329–342.
Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.
Thiébaux, S., Gretton, C., Slaney, J.K., Price, D., Kabanza, F., et al., 2006. Decision-theoretic planning with non-Markovian rewards. J. Artif. Intell. Res. 25, 17–74.
Tiwari, A., 2008. Abstractions for hybrid systems. Formal Methods Syst. Des. 32, 57–83.
Van Kranenburg, R., Anzelmo, E., Bassi, A., Caprio, D., Dodson, S., Ratto, M., 2008. The internet of things. A critique of ambient technology and the all-seeing network of RFID, Network Notebooks 2.
Wang, Y., de Silva, C.W., 2008. A machine-learning approach to multi-robot coordination. Eng. Appl. Artif. Intell. 21 (3), 470–484.
Werbos, P.J., 1991. A menu of designs for reinforcement learning over time. In: Miller, W.T., Sutton, R.S., Werbos, P.J. (Eds.), Neural Networks for Control. MIT Press, Cambridge, MA, pp. 67–95.
Whittle, J., Kwan, R., Saboo, J., 2005. From scenarios to code: an air traffic control case study. Softw. Syst. Model. 4 (1), 71–93.
Yokomori, T., 2003. Polynomial-time identification of very simple grammars from positive data. Theor. Comput. Sci. 298 (1), 179–206.
Yoshinaka, R., 2011. Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. Theor. Comput. Sci. 412 (19), 1821–1831.