Formal Methods with a Touch of Magic

Parand Alizadeh Alamdari
Sharif University of Technology

Guy Avni, Thomas A. Henzinger, and Anna Lukina
IST Austria

***Abstract*—Machine learning and formal methods have complimentary benefits and drawbacks. In this work, we address the controller-design problem with a combination of techniques from both fields. The use of black-box neural networks in deep reinforcement learning (deep RL) poses a challenge for such a combination. Instead of reasoning formally about the output of deep RL, which we call the *wizard*, we extract from it a decision-tree based model, which we refer to as the *magic book*. Using the extracted model as an intermediary, we are able to handle problems that are infeasible for either deep RL or formal methods by themselves. First, we suggest, for the first time, a synthesis procedure that is based on a magic book. We synthesize a stand-alone correct-by-design controller that enjoys the favorable performance of RL. Second, we incorporate a magic book in a bounded model checking (BMC) procedure. BMC allows us to find numerous traces of the plant under the control of the wizard, which a user can use to increase the trustworthiness of the wizard and direct further training.**

I. Introduction

Machine-learning techniques and, in particular, the use of neural networks (NNs), are exploding in popularity and becoming a vital part of the development of many technologies. There is a challenge, however, in deploying systems that use trained components, which are inherently black-box. For a system to be used by a human, it must be trustworthy: provably correct, or predictable, at the least. Current trained systems lack either of these properties.

In this work, we focus on the controller-design problem. Abstractly speaking, a controller is a device that interacts with a plant. At each time step, the plant outputs its state and the controller feeds back an action. Combining techniques from both formal methods and machine learning is especially appealing in the controller-design problem since both correctness and performance are critical.

Reinforcement learning (RL) is the main machine-learning tool for designing controllers. The RL approach is based on "trial and error": the agent randomly explores its environment, receives rewards and learns from experience how to maximize them. RL has made a quantum leap in terms of scalability since the recent introduction of NNs into the approach, termed *deep RL* [1]. We call the output of deep RL the *wizard*: it optimizes plant performance but, since it is a NN, it does not reveal its decision procedure. More importantly, there are no guarantees on the wizard and it can behave unexpectedly and even incorrectly.

Reasoning about systems that use NNs poses a challenge for formal methods. First, in terms of scalability (NNs tend to be large), and second, the operations that NNs depend on are challenging for formal methods tools, namely NNs use numerical rather than Boolean operations and *ReLu* neurons use the `max` operator, which SMT tools struggle with.

We propose a novel approach based on extracting a decision-tree-based model from the wizard, which approximates its operation and is intended to reveal its decision-making process. Hence, we refer to it as the *magic book*. Our requirements for the magic book are that it is (1) simple enough for formal methods to use, and (2) a good approximation of the NN.

Extracting decision-tree-based models that approximate a complicated function is an established practice [2]. The assumption that allows this extraction to work is that a NN contains substantial redundancy. During training, the NN "learns" heuristics that it uses to optimize plant performance. The heuristics can be compactly captured in a small model, e.g., in a decision-tree. This assumption has led, for example, to attempts of distilling knowledge from a trained NN to a second NN during its training [3], [4], and of minimizing NNs (e.g., [5]). The extraction of a simple model is especially common in *explainable AI* (XAI) [6], where the goal is to explain the operation of a learned system to a human user.

We use the tree-based magic book to solve problems that are infeasible both for deep RL and for formal methods alone. Specifically, we illustrate the magic book's benefit in two approaches for designing controllers as we elaborate below.

Reactive synthesis [7] is a formal approach to design controllers. The input is a qualitative specification and the output is a correct-by-design controller. The fact that the controller is provably correct, is the strength of synthesis. A first weakness of traditional synthesis is that it is purely qualitative and specifications cannot naturally express quantitative performance. There is a recent surge of quantitative approaches to synthesis (e.g., [8], [9], [10]). However, these approaches suffer from other weaknesses of synthesis: deep RL vastly outperforms synthesis in terms of scalability. Also, in the average-case, RL-based controllers beat synthesized controllers since the goal in synthesis is to maximize worst-case performance.

Synthesis is often reduced to solving a two-player graph game; Player 1 represents the controller and Player 2 represents the plant. In each step, Player 2 reveals the current state $\bar{s}$ of the plant and Player 1 responds by choosing an action. In our construction, when Player 2 chooses $\bar{s}$, we extract from the magic book the action a that is taken at $\bar{s}$. Player 1's action then depends on a as we elaborate below. The construction of the game arena thus depends on the magic book, and using the wizard instead is infeasible.

We present a novel approach for introducing performance

https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_21

considerations into reactive synthesis. We synthesize a controller that satisfies a given qualitative specification while following the magic book as closely as possible. We formalize the later as a quantitative objective: whenever Player 1 agrees with the choice of action suggested by Player 2, he receives a reward, and the goal is to maximize rewards. Since the magic book is a proxy for the RL-generated wizard, we obtain the best of both worlds: a provably correct controller that enjoys the high average-case performance of RL. In our experiments, we synthesize a controller for a taxi that travels on a grid for the specification "visit a gas station every t steps" while following advice from a wizard that is trained to collect as many passengers as possible in a given time frame.

In a second application, we use a magic book to relax the adversarial assumption on the environment in a multi-agent setting. We are thus able to synthesize controllers for specifications that are otherwise *unrealizable*, i.e., for which traditional synthesis does not return any controller. Our goal is to synthesize a controller for an agent that interacts with an environment that consists of other agents. Instead of modeling the other agents as adversarial, we assume that they operate according to a magic book. This restricts their possible actions and regains realizability. For example, suppose a taxi that is out of our control, shares a network of roads with a bus, under our control. Our goal is to synthesize a controller that guarantees that the bus travels between two stations without crashing into a taxi. While an adversarial taxi can block the bus, by assuming that the taxi operates according to a magic book, we limit Player 2's action in the game and find a winning Player 1 strategy that corresponds to a correct controller.

Bounded model checking [11] (BMC) is an established technique to find bounded traces of a system that satisfy a given specification. In a second approach to the controller-design problem, we use BMC as an XAI tool to increase the trustworthiness of a wizard before outputting it as the controller of the plant. We rely on BMC to find (many) traces of the plant under the control of the wizard that are tedious to find manually.

We solve BMC by constructing an SMT program that intuitively simulates the operation of the plant under the control of the magic book rather than under the control of the wizard. The traces we find witness the magic book. A disadvantage of the approach is that it is not sound (see Remark 3). The advantage is that the reduction from BMC to SMT is simple and leads to a significant performance gain: in our experiments, we use the standard SMT solver Z3 [12] to extract thousands of witnesses within minutes, whereas Z3 is incapable of solving extremely modest wizard-based BMC instances. Before outputting a trace, we perform a secondary test to check that it witnesses the wizard as well. In our experiments, we find that many traces are indeed shared between the two. Thus, our procedure efficiently finds numerous traces of the plant under the control of the wizard.

A first application of BMC is in verification; namely, we find counterexamples for a given specification. For example, when controlling a taxi, a violation of a liveness property is an infinite loop in which no passenger is collected. We find it more appealing to use BMC as an XAI tool. For example, BMC allows us to find "suspicious" traces that are not necessarily incorrect; e.g., when controlling a taxi, a passenger that is not closest is collected first. Individual traces can serve as explanations. Alternatively, we use BMC's ability to find many traces and gather a large dataset. We extract a small human-interpretable model from the dataset that attempts to explain the wizard's decision-making procedure. For example, the model serves as an answer to the question: when does the wizard prefer collecting a passenger that is not closest?

A. Related work

We compare our synthesis approach to *shielding* [13], [14], which adds guarantees to a learned controller at runtime by monitoring the wizard and correcting its actions. Unlike shielding, the magic book allows us to open up the black-box wizard, which, for example, enables our controller to cross an obstacle that was not present in training, a task that is inherently impossible for a shield-based controller. A second key difference is that we produce stand-alone controllers whereas a shield-based approach needs to execute the NN wizard in each step. Our method is thus preferable in settings where running a NN is costly, e.g., embedded systems or real time systems.

To the best of our knowledge, synthesis in combination with a magic book was never studied. Previously, finding counterexamples for tree-based controllers that are extracted from NN controllers was studied in [15] and [16]. The ultimate goal in those works is to output a correct tree-based controller. A first weakness of this approach is that, since both wizard and magic book are trained, they exhibit many correctness violations. We believe that repairing them manually while maintaining high performance is a challenging task. Our synthesis procedure assists in automating this procedure. Second, in some cases, a designer would prefer to use a NN controller rather than a tree-based one since NNs tend to generalize better than tree-based models. Hence, we promote the use of BMC for XAI to increase the trustworthiness of the wizard. Finally, the case studies the authors demonstrate are different from ours, thus they strengthen the claim that a tree-based classifier extraction is not specific to our domain rather it is a general concept.

A specialized wizard-based BMC tool was recently shown in [17], thus unlike our approach, there is no need to check that the output trace is also a witness for the wizard. More importantly, their method is "sound": if their method terminates without finding a counterexample for bound $\ell \in \mathbb{N}$, then there is indeed no violation of length ℓ. Beyond the disadvantages listed above, the main disadvantage of their approach is scalability, which is not clear in the paper. As we describe in the experiments section, our experience is that a wizard-based BMC implemented in Z3 does not scale.

Our BMC procedure finds traces that witness a temporal behavior of the plant. This is very different from finding *adversarial examples*, which are inputs slightly perturbed so

that to lead to a different output. Finding adversarial examples and verifying robustness have attracted considerable attention in NNs (for example, [18], [19], [20]) as well as in random-forest classifiers (e.g., [21], [22]).

Somewhat similar in spirit to our approach is applying *program synthesis* to extract a program from a NN [23], [24], [25], which, similar to the role of the magic book, is an alternative small model for application of formal methods. The main goal in these papers is to extract a magic book ("program", in their terminology) from a wizard, verify its correctness and use it as the controller for the plant. A key difference from our synthesis approach is that their wizard is trained to satisfy the specification and the challenge is to devise a good approximation for the wizard. Our wizard, however, is trained without consideration for the specification; e.g., neither the gas station nor the obstacles mentioned above are present in training. The challenge is to incorporate the wizard in synthesis to gain both performance and correctness.

Decision trees were previously used to represent, in a succinct, verifiable and explainable manner, a strategy for a controller (e.g., [26], [27]). The challenge here is to construct a concise controller from a given policy, similar to the works above. A second difference is that the policy is given explicitly and is obtained from an explicit solution to the MDP or game, hence scalability is an inherent limitation of this approach.

Finally, examples of other combinations of RL with synthesis include works that run an online version of RL (see [28] and references therein), an execution of RL restricted to correct traces [29], and RL with safety specifications [30].

II. Preliminaries

a) Plant and controller: We formalize the interaction between a controller and a plant. The plant is modelled as a *Markov decision process* (MDP) which is $\mathcal{M} = \left(S, \bar{s}_0, A, R, p\right)$, where S is a finite set of states, $\bar{s}_0 \in S$ is an initial configuration of the state, A is a finite collection of actions, $R : S \to \mathbb{R}$ is a reward provided in each state, and $p : S \times A \to [0,1]^S$ is a probabilistic transition function that, given a state and an action, produces a probability distribution over states.

Example 1. *Our running example throughout the paper is a taxi that travels on an $n \times n$ grid and collects passengers. Whenever a passenger is collected, it re-appears in a random location. A state of the plant contains the locations of the taxi and the passengers, thus it is a tuple $\bar{s} = \left(p_0, p_1, \ldots, p_k\right)$, where for $0 \leq i \leq k$, the pair $p_i = \left(x_i, y_i\right)$ is a position on the grid, p_0 is the position of the taxi, and p_i is the position of Passenger i. The set of actions is $A = \{\texttt{up}, \texttt{right}, \texttt{down}, \texttt{left}\}$. The transitions of $\mathcal{M}$ are largely deterministic: given an action $a \in A$, we obtain the updated state $\bar{s}'$ by updating the position of the taxi deterministically, and if the taxi collects a passenger, i.e., $p'_0 = p_i$, for some $1 \leq i \leq k$, then the new position of Passenger i is chosen uniformly at random.*

The controller is a *policy*, which prescribes which action to take given the history of visited states, thus it is a function $\pi : S^* \to A$. A policy is *positional* if the action that it prescribes depends only on the current position, thus it is a function $\pi : S \to A$. We are interested in finding an optimal and correct policy as we define below.

b) Qualitative correctness: We consider a strong notion of qualitative correctness that disregards probabilistic events, often called *surely* correctness. A specification is $\Omega \subseteq S^\omega$. We define the *support* of p at $\bar{s}$ given $a \in A$ as $\mathrm{supp}(\bar{s}, a) = \{\bar{s}' : p\left(\bar{s}' \mid \bar{s}, a\right) > 0\}$ and, for a policy π, we define the support of π to be $\mathrm{supp}_\pi(\bar{s}) = \mathrm{supp}(\bar{s}, \pi(\bar{s}))$. We define the *surely language* of $\mathcal{M}$ w.r.t. π, denoted $L_\pi(\mathcal{M})$. A run $\sigma = \sigma_1, \sigma_2, \ldots \in S^\omega$ is in $L_\pi(\mathcal{M})$ iff we have $\sigma_1 = \bar{s}_0$ and for every $i \geq 1$, we have $\sigma_{i+1} \in \mathrm{supp}_\pi(\sigma_i)$, where $a_i = \pi(\sigma_1, \ldots, \sigma_i)$. We say that π is surely-correct for plant $\mathcal{M}$ w.r.t. a specification $\Omega \subseteq S^\omega$ iff it allows only correct runs of $\mathcal{M}$, thus $L_\pi(\mathcal{M}) \subseteq \Omega$.

c) Quantitative performance and deep reinforcement learning: The goal of reinforcement learning (RL) is to find a policy in an MDP that maximizes the expected reward [31]. In a finite MDP $\mathcal{M}$, the state at a time step $t \in \mathbb{N}$ is a random variable, denoted s_t. Each time step entails a reward, which is also a random variable, denoted r_t. The probability that s_t and r_t get particular values depends solely on the previous state and action. Formally, for an initial state $\bar{s}_0 \in S$, we define $\Pr[s_0 = \bar{s}_0] = 1$, and for $\bar{s}', \bar{s} \in S$ and $a \in A$, we have $\Pr\left[s_t = \bar{s}', r_t = R(s) \mid s_{t-1} = \bar{s}, a_{t-1} = a\right] = p\left(\bar{s}' \mid \bar{s}, a\right)$. We consider *discounted rewards*. Let $\gamma \in (0,1)$ be a discount factor. The *expected reward* that a policy π ensures starting at state $\bar{s} \in S$ is $Rew_\pi(\bar{s}) = \sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is defined w.r.t. $\bar{s}$ as in the above. The goal is to find the optimal policy π^* that attains $\sup_\pi Rew_\pi(\bar{s}_0)$.

We consider the *Q-learning* algorithm for solving MDPs, which relies on a function $Q : S \times A \to \mathbb{R}$ such that $Q(\bar{s}, a)$ represents the expected value under the assumption that the initial state is $\bar{s}$ and the first action to be taken is a, thus $Q(\bar{s}, a) = R(\bar{s}) + \gamma \cdot \sum_{\bar{s}'} p\left(\bar{s}' \mid \bar{s}, a\right) \cdot Rew_{\pi^*}(\bar{s}')$. Clearly, given the function Q, one can obtain an optimal positional policy π^*, by defining $\pi^*(\bar{s}) = \arg\max_a Q(\bar{s}, a)$, for every state $\bar{s} \in S$. In Q-learning, the Q function is estimated and iteratively refined using the Bellman equation.

Traditional implementations of Q-learning assume that the MDP is represented explicitly. Deep RL [1] implements the Q-learning algorithm using a symbolic representation of the MDP as a NN. The NN takes as input a state $\bar{s}$ and outputs for each $a \in A$, an estimate of $Q(\bar{s}, a)$. The technical challenge in deep RL is that it combines training of the NN with estimating the Q function. We call the NN that deep RL outputs the *wizard*. Even though deep RL does not provide any guarantees on the wizard, in practice it has shown remarkable success.

d) Magic books from decision-tree-based classifiers: Recall that the output of deep RL is a positional function that is represented by a NN $\textsc{Wiz} : S \to A$. We are interested in extracting a small function MB of the same type that approximates Wiz well. We use *decision-tree based classifiers* as our model of choice for MB. Each internal node v of a decision tree is labeled with a predicate $\varphi(v)$ over S and each leaf is labeled with an action in A. A plant state $\bar{s}$ gives rise

to a unique path in a decision tree $\mathcal{T}$, denoted path$(\mathcal{T}, \bar{s})$, in the expected manner. The first node is the root. Upon visiting an internal node v, the next node in path$(\mathcal{T}, \bar{s})$ depends on the satisfaction value of $\varphi(v)(\bar{s})$. Suppose $\varphi_1, \ldots, \varphi_n$ is the sequence of predicates traversed by a path $\eta = \text{path}(\mathcal{T}, \bar{s})$, we use pred$(\eta)$ to denote $\varphi_1 \wedge \ldots \wedge \varphi_n$. Thus, for every $\bar{s}' \in S$ we have $\eta = \text{path}(\mathcal{T}, \bar{s}')$ iff $\bar{s}'$ satisfies pred(η). When path$(\mathcal{T}, \bar{s})$ ends in a leaf labeled $a \in A$, we say that the tree *votes* for a. A *forest* contains several trees. On input $\bar{s} \in S$, each tree votes for an action and the action receiving most votes is output.

To obtain MB from WIZ, we first execute WIZ with the plant on a considerable number T of steps, to collect pairs of the form $\big(\bar{s}_t, \text{WIZ}(\bar{s}_t)\big)$, for $t \in \{0, \ldots, T\}$, where $\bar{s}_t$ is the system state at time t and T is chosen to maximize model's F1 score. We then employ standard techniques on this dataset to construct either a decision tree, or a forest of decision trees using the state-of-the-art *random forest* [32] or *extreme gradient boosting* [33] techniques.

Remark 1. *One might wonder whether it is possible to obtain a decision tree (magic book) directly from RL, thus making the wizard obsolete. While there were attempts at using decision trees as the underlying reward approximation in RL (e.g., [34]), the approach has inherent limitations (see details in https://bit.ly/30WBA1i). Moreover, decision trees are popular data structures that are often preferred to NNs since they are simpler, easier to interpret, and have less parameters to tune. Still, the literature on deepRL overshadows the literature on RL with decision trees. Also, the choice to extract a decision tree from a NN rather than training a decision tree directly was also made in [15], [16], and their case studies differ from ours, strengthening the claim that this approach is general. Finally, we note that even if the magic book is obtained directly from RL, it does not solve the challenges we address in our synthesis and BMC procedures.*

III. Synthesis with a Touch of Magic

Our primary goal in this section is to automatically construct a correct controller and performance is a secondary consideration. We incorporate a magic book into synthesis and illustrate two applications of the constructions that are infeasible without a magic book.

A. Constructing a game

Synthesis is often reduced to a two-player graph game (see [35]). In this section, we describe a construction of a game *arena* that is based on a magic book and in the next sections we complete the construction by describing the players' objectives and illustrate applications. In the traditional game, Player 2 represents the environment and in each turn, he reveals the current location of the plant. Player 1, who represents the controller, answers with an action. A strategy for Player 1 corresponds to a policy (controller) since, given the history of observed plant states, it prescribes which action to feed in to the plant next. The traditional goal is to find a Player 1 strategy that guarantees that a given specification Ω is satisfied no matter how Player 2 plays. Traditional synthesis is purely qualitative; namely, it returns some correct policy with no consideration to its performance. When no correct controller exists, we say that Ω is *un-realizable*.

Formally, a graph game is played on an arena $\big(V, \Xi_1, \Xi_2, \delta\big)$, where V is a set of vertices, for $i \in \{1, 2\}$, Player i's possible actions are Ξ_i, and $\delta : V \times \Xi_1 \times \Xi_2 \to V$ is a deterministic transition function. The game proceeds by placing a token on a vertex in V. When the token is placed on $v \in V$, Player 2 moves first and chooses $\xi_2 \in \Xi_2$. Then, Player 1 chooses $\xi_1 \in \Xi_1$ and the token proceeds to $\delta(v, \xi_1, \xi_2)$. In games, rather than using the term "policy", we use the term *strategy*. Two strategies f and g for the two players and an initial vertex induce a unique infinite play, which we denote by $play(f, g)$, where for ease of notation we omit the initial vertex.

We describe our construction in which the roles of the players is slightly altered. Consider a plant $\mathcal{M}$ with state space S and actions A. The arena of our synthesis game is based on two abstractions Γ_1 and Γ_2 of S. While we assume Γ_1 is provided by a user, the partition Γ_2 is extracted from the magic book. The arena is $\mathcal{A} = \big(\Gamma_1, A, \Gamma_2, \delta\big)$, where δ is defined below. Suppose that the token is placed on $\gamma_1 \in \Gamma_1$ (see Fig. 1). Intuitively, the actual location of the plant is a state $\bar{s} \in S$ with $\bar{s} \in \gamma_1$. Player 2 moves first and chooses a set $\gamma_2 \in \Gamma_2$ such that $\gamma_1 \cap \gamma_2 \neq \emptyset$. Intuitively, a Player 2 action reveals that the actual state of the plant is in $\gamma_1 \cap \gamma_2$. Player 1 reacts by choosing an action $a \in A$. We denote by supp$(\gamma_1 \cap \gamma_2, a)$ the set of possible next locations the plant can be in, thus supp$(\gamma_1 \cap \gamma_2, a) = \{\bar{s}' : \exists \bar{s} \in \gamma_1 \cap \gamma_2$ with $\bar{s}' \in \text{supp}(\bar{s}, a)\}$. Then, the next state in the game according to δ is the minimal-sized set $\gamma_1' \in \Gamma_1$ such that supp$(\gamma_1 \cap \gamma_2, a) \subseteq \gamma_1'$.

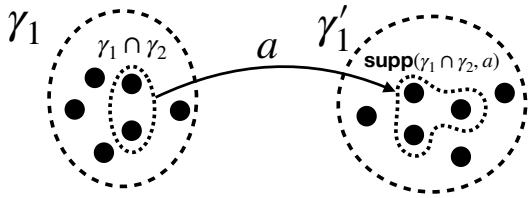

Fig. 1: A transition between two abstract states $\gamma_1, \gamma_1' \in \Gamma_1$; black dots represent states in S. For every $\bar{s} \in \gamma_1 \cap \gamma_2$, we have MB$(\bar{s}) = a$.

Suppose for ease of presentation that the magic book is a decision tree $\mathcal{T}$, and the construction easily generalizes to forests. Recall that a state $\bar{s} \in S$ produces a unique path $\eta = \text{path}(\mathcal{T}, \bar{s})$, which corresponds to sequence of predicates $\varphi_1, \ldots, \varphi_n$, and pred$(\eta) = \bigwedge_{1 \leq i \leq n} \varphi_i$. We define $\Gamma_2 = \{\text{pred}(\eta) : \eta$ is a path in $\mathcal{T}\}$. Let η be a path in $\mathcal{T}$ and $\gamma_2 \in \Gamma_2$ the corresponding predicates. For ease of notation, we abuse notation and refer to γ_2 as the set of states in S who produce the path η in $\mathcal{T}$. An immediate consequence of the construction is the following.

Lemma 1. *For every $\gamma_2 \in \Gamma_2$ there is $a \in A$ such that* MB$(\bar{s}) = a$*, for all $\bar{s} \in \gamma_2$.*

In the following lemma we formalize the intuition that Player 2 over-approximates the plant. It is not hard, given a Player 1 strategy f, to obtain a policy $\pi(f)$ that follows it.

For $\bar{s} \in S$, we use $\gamma_1(\bar{s}) \in \Gamma_1$ and $\gamma_2(\bar{s}) \in \Gamma_2$ to denote the unique abstract set that $\bar{s}$ belongs to.

Lemma 2. *Let f be a Player 1 strategy. Consider a trace $\sigma = \sigma_1, \sigma_2, \ldots \in L_{\pi(f)}(\mathcal{M})$. Then, there is a Player 2 strategy g such that $play(f, g) = \gamma_1(\sigma_1), \gamma_1(\sigma_2), \ldots$.*

Proof. We define g inductively so that for every $n \geq 1$, the n-th vertex of $play(f, g)$ is $\gamma_1(\sigma_n)$. Suppose the invariant holds for σ_n. Player 2 chooses $\gamma_2(\sigma_n)$. The definition of δ implies that the invariant is maintained, thus $\sigma_{n+1} \in \delta\big(\gamma_1(\sigma_n), f(\gamma_1(\sigma_n), \gamma_2(\sigma_n))\big)$. □

We note that the converse of Lemma 2 is not necessarily correct, thus Player 2 strictly over-approximates the plant. Indeed, suppose that the token is placed on γ_1, Player 2 chooses γ_2, Player 1 chooses $a \in A$, and the token proceeds to γ_1'. Intuitively, the plant state was in $\gamma_1 \cap \gamma_2$ and thus should now be in $\text{supp}(\gamma_1 \cap \gamma_2, a)$. In the subsequent move, however, Player 2 is allowed to choose any γ_2' with $\gamma_1' \cap \gamma_2' \neq \emptyset$, even one that does not intersect $\text{supp}(\gamma_1 \cap \gamma_2, a)$.

B. Following expert advice

In this section, we abstain from solving the problem of finding a correct and optimal controller; a problem that is computationally hard for explicit systems, not to mention symbolically-represented systems like the ones we consider. Instead, in order to add performance consideration to synthesis, we think of the wizard as an authority in terms of performance and solve the (hopefully simpler) problem of constructing a correct controller that follows the wizard's actions as closely as possible. We use the magic book as a proxy for the wizard and assume that following its actions most of the time results in favorable performance.

The game arena is constructed as in the previous section. Player 1's goal is to ensure that a given specification Ω is satisfied while optimizing a quantitative objective that we use to formalize the notion of "following the magic book". For simplicity, we consider finite paths, thus $\Omega \subseteq \Gamma_1^*$, and the definitions can be generalized to infinite plays. By Lem. 1, every Player 2 action $\gamma_2 \in \Gamma_2$ corresponds to a unique action in A, which we denote by $a(\gamma_2) \in A$. We think of Player 2 as "suggesting" the action $a(\gamma_2)$ since for every $\bar{s} \in \gamma_2$, we have $\text{MB}(\bar{s}) = a(\gamma_2)$. To motivate Player 1 to use $a(\gamma_2)$, when he "accepts" the suggestion and chooses the same action, he obtains a reward of 1 and otherwise he obtains no reward. Then, Player 1's goal in the game is to maximize the sum of rewards that he obtains.

We formalize the guarantees of the controller $\pi(f)$ that we synthesize w.r.t. an optimal strategy f for Player 1. Intuitively, the payoff that f guarantees in the game is a lower on the number of times $\pi(f)$ agrees with the magic book in any trace of the plant. Let f and g be two strategies for the two players. We use $\texttt{Score}(f, g)$ to denote the payoff of Player 1 in the game. When $play(f, g) \notin \Omega$, we set $\texttt{Score}(f, g) = \infty$, thus Player 1 first tries to ensure that Ω holds. If $play(f, g) \in \Omega$, the score is the sum of rewards in $play(f, g)$. We assign a score to $\pi(f)$ in a path-based manner. Let $\sigma = \sigma_1, \ldots, \sigma_n \in L_{\pi(f)}(\mathcal{M})$. For every $1 \leq i \leq n$, we issue a reward of 1 if $\pi(f)(\sigma_i) = \text{MB}(\sigma_i)$, and we denote by $\texttt{Agree}(\pi(f), \sigma)$, the sum of rewards, which represents the sum of states in which $\pi(f)$ agrees with MB throughout σ. The following theorem follows from Lem. 2.

Theorem 1. *Let f^* be a strategy that achieves $x^* = \max_f \min_g \texttt{Score}(f, g)$. If $x^* < \infty$, then $\pi(f)$ is correct w.r.t. Ω. Moreover, for every $\sigma \in L_{\pi(f^*)}(\mathcal{M})$ we have $\texttt{Agree}(\pi(f^*), \sigma) \geq x^*$.*

C. Multi-agent synthesis

In this section, we design a controller in a multi-agent setting, where traditional synthesis is unrealizable and thus does not return any controller.

For ease of presentation, we focus on two agents, and the construction can be generalized to more agents in a straightforward manner. We assume that the set of actions A is partitioned between the two agents, thus $A = A_1 \times A_2$. In each step, the players simultaneously select actions, where for $i \in \{1, 2\}$, Player i selects an action in $a_i \in A_i$. As before, the joint action determines a probability distribution on the next state according to δ. Our goal is to find a controller for Agent 1 that satisfies a given specification Ω no matter how Player 2 plays.

Example 2. *Suppose that the grid has two means of transportation: a bus (Agent 1) and a taxi (Agent 2). We are interested in synthesizing a bus controller for the specification "travel between two stations while not hitting the taxi". If one models the taxi as an adversary, the specification is clearly not realizable: the taxi parks in one of the targets so that the bus cannot visit it without crashing into the taxi.*

We assume that Agent 2 is operating according to a magic book. As in the previous section, we require an abstraction Γ_1 such that $\Omega \subseteq \Gamma_1^\omega$ and the abstraction Γ_2 is obtained from the magic book. We construct a game arena as in Section III-A and Player 1 wins an infinite play iff it satisfies Ω.

The way the magic book is employed here is that it restricts the possible actions that Player 2 can take. Going back to the taxi and bus example, at a state $\bar{s} \in S$, Player 2 essentially chooses how to move the taxi. Suppose the token is placed on $\gamma_1 \in \Gamma_1$. Player 2 cannot choose to move the taxi in any direction; indeed, he can choose $a_2 \in A_2$ only when there is a state $\bar{s} \in \gamma_1$ such that $\text{MB}(\bar{s}) = a_2$. The following theorem is an immediate consequence of Lem. 2.

Theorem 2. *Let f be a* winning *strategy: for every g, $play(f, g)$ satisfies Ω. Then, $L_\pi(f)(\mathcal{M}) \subseteq \Omega$.*

In Remark 3 we discuss the guarantees on the magic book that are needed to assume that Agent 2 operates according to a wizard rather than a magic book.

IV. BMC Based on Magic Books

In this section, we describe a bounded-model-checking (BMC) [11] procedure that is based on a tree-based magic

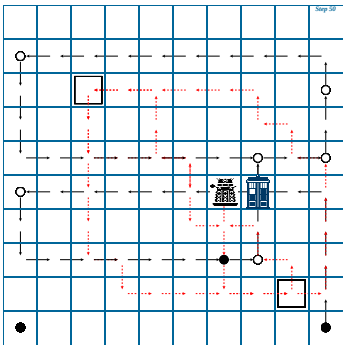

Fig. 2: Snapshot of step 50 of the simulation. A bus controlled by a synthesized controller (depicted in red dashed arrows and travelling between two square stations) shares the grid with a taxi that is controlled by a magic-book (depicted with black arrows).

book. We use our procedure in verification and as an explainability tool to increase the trustworthiness of the wizard before outputting it as the controller for the plant.

Definition 1 (Bounded model checking)**.** *Given a plant $\mathcal{M}$ with state space S, a specification Ω, a bound $\ell \in \mathbb{N}$, and a policy π, output a run of length ℓ in $L_\pi(\mathcal{M}) \cap \Omega$ if one exists.*

BMC reduces to the satisfiability problem for *satisfiability modulo theories* (SMT), where the goal is, given a set of constraints over a set of variables X, either find a satisfying assignment to X or return that none exists. We are interested in solving BMC for wizards, i.e., finding a path in $L_{\text{WIZ}}(\mathcal{M}) \cap \Omega$. However, as can be seen in the proof of Thm. 3 below, the SMT program needs to simulate the execution of the wizard, thus it becomes both large and challenging (due to the `max` operator) for standard SMT solvers. Instead, we solve BMC for magic books to find a path $\eta \in L_{\text{MB}}(\mathcal{M}) \cap \Omega$. Since MB is a good approximation for WIZ, we often have $\eta \in L_{\text{WIZ}}(\mathcal{M}) \cap \Omega$.

Theorem 3. *BMC reduces to SMT. Specifically, given a plant $\mathcal{M}$ with states S, a specification $\Omega \subseteq S^*$, a policy π given as a tree-based magic book, and a bound ℓ, there is an SMT formula whose satisfying assignments correspond to paths of length ℓ in $L_\pi(\mathcal{M}) \cap \Omega$.*

Proof. The first steps of the reduction are standard. Consider a policy π and a bound $\ell \in \mathbb{N}$. The variables consist of *state variables* $X_0, \ldots, X_\ell$ and *action variables* $Y_0, \ldots, Y_{\ell-1}$. We add constraints so that, for a satisfying assignment α, for $0 \le i \le \ell$, each $\alpha(X_i)$ corresponds to a state in S, and for $0 \le i \le \ell - 1$, each $\alpha(Y_i)$ corresponds to an action in A. Moreover, for $0 \le i \le \ell - 1$, the constraints ensure that $\alpha(X_{i+1}) \in \text{supp}(\alpha(X_i), \alpha(Y_i))$, thus we obtain a path in $L_\pi(\mathcal{M})$.

We consider a specification Ω that can be represented as an SMT constraint over $X_0, \ldots, X_\ell$ and add constraints so that the path we find is in $L_\pi(\mathcal{M}) \cap \Omega$.

The missing component from this construction ensures that the action $\alpha(Y^i)$ is indeed the action that π selects at state $\alpha(X^i)$. For that, we need to simulate the operation of π using constraints. Suppose first that π is represented using a decision tree $\mathcal{T}$. For a path η in $\mathcal{T}$, recall that $\text{pred}(\eta)$ is the predicate $\varphi_1 \wedge \ldots \wedge \varphi_n$ that is satisfied by every state $\bar{s} \in S$ such that $\text{path}(\mathcal{T}, \bar{s}) = \eta$. Moreover, recall that each φ_j is a predicate over S. For $0 \le i < \ell$, we create a copy of $\text{pred}(\eta)$ using the variables X_i so that it is satisfied iff $\alpha(X_i)$ satisfies $\text{pred}(\eta)$. For $a \in A$, let $\text{paths}(\mathcal{T}, a)$ denote the set of paths in $\mathcal{T}$ that end in the action a. We add a constraint that states that if $\bigvee_{\eta \in \text{paths}(\mathcal{T},a)} \text{pred}(\eta)$ is true at time i, then $\alpha(Y_i) = a$. Finally, when MB is a forest, we need to count the number of trees that vote for each action and set $\alpha(Y_i)$ to equal the action with the highest count. □

Remark 2. (The size of the SMT program). *In the construction in Theorem 3, as is standard in BMC, we use roughly ℓ copies of $\mathcal{M}$, where the size of each copy depends on the representation size of $\mathcal{M}$. In addition, we need a constraint that represents Ω, which in our examples, is of size $O(\ell)$. The main bottleneck are the constraints that represent π. Each path appears exactly once in a constraint, and we use $\ell + 1$ copies of π, thus the total size of these constraints is $O(\ell \cdot |\pi|)$, where $|\pi|$ is the number of paths in the trees in the forest.*

Example 3. *Recall the description of the plant in Example 1 in which a taxi travels in a grid. We illustrate how to simulate the plant using an SMT program. A state at time i is a $2 \cdot (k+1)$ tuple of variables $\left(x_0^i, y_0^i, \ldots, x_k^i, y_k^i\right)$ that take integer values in $\{0, \ldots, n\}$. The position of the taxi at time i is (x_0^i, y_0^i) and the position of Passenger j is (x_j^i, y_j^i). The transition function is represented using constraints. For example, the constraint $\left(Y_i = \texttt{up}\right) \rightarrow \left((x_0^{i+1} = x_0^i) \wedge (y_0^{i+1} = y_0^i + 1)\right)$ means that when the action `up` is taken, the taxi moves one step up. The constraint $\neg\left((x_0^{i+1} = x_j^i) \wedge (y_0^{i+1} = y_j^i)\right) \rightarrow \left((x_j^{i+1} = x_j^i) \wedge (y_j^{i+1} = y_j^i)\right)$ means that if Passenger j is not collected by the taxi at time $i + 1$, its location should not change. A key point is that when Passenger j is collected, we do not constrain his new location, thus we replace the randomness in $\mathcal{M}$ with nondeterminism.*

a) Verification: In verification, our goal is to find violations of the wizard for a given specification.

Example 4. *We show how to express the specification "the taxi never enters a loop in which no passenger is collected" as an SMT constraint based on the construction in Example 3. We simplify slightly and use the constraint $\left(x_0^\ell = x_0^0 \wedge y_0^\ell = y_0^0\right)$ that means that the taxi returns to its initial position to close a cycle at the end of the trace. We add a second constraint $\bigwedge_{1 \le j \le k} \bigwedge_{1 \le i \le \ell} \left(x_j^0 = x_j^i \wedge y_j^0 = y_j^i\right)$ that means that all passengers stay in their original position throughout the trace. In Fig. 3 (right), we depict a lasso-shaped trace that witnesses a violation of this property.*

Remark 3 (Soundness)**.** *The benefit of using magic books is scalability, and the draw-back is soundness. For example, when the SMT formula is unsatisfiable for a bound $\ell \in \mathbb{N}$, this only means that there are no violations of the* magic book *of length ℓ, and there can still be a violation of the wizard. To regain soundness we would need guarantees on the*

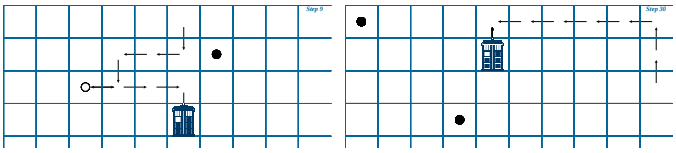

Fig. 3: Examples found using BMC. Left: a snapshot of step 9 of the simulation showing the closest passenger was not collected first. The passenger collected first is shown as a hollow circle. The passengers not yet collected are shown as filled black circles. Right: a snapshot of step 30 of the simulation of a "lasso"-shaped trace of the taxi that entered a loop without collecting any passengers.

relation between the magic book and the wizard. An example of a guarantee is that the two functions coincide, thus for every state $\bar{s} \in S$, we have $\text{WIZ}(\bar{s}) = \text{MB}(\bar{s})$. *However, if at all possible, we expect such a strong guarantee to come at the expense of a huge magic book, thus bringing us back to square one. We are more optimistic that one can find small magic books with approximation guarantees. For example, one can define a magic book as a function* $\text{MB} : S \to 2^A$ *that "suggests" a set of actions rather than only one, and require that for every state $\bar{s} \in S$, we have* $\text{WIZ}(\bar{s}) \in \text{MB}(\bar{s})$. *Such guarantees suffice to regain soundness both in BMC and for the synthesis application in Section III-C. We leave for future work obtaining such magic books.*

b) Explainability: We illustrate how BMC can be used as an XAI tool. BMC allows us to find corner-case traces that are hard to find in a manual simulation and the individual traces can serve as explanations. For example, in Fig. 3 (left), we depict a trace that is obtained using BMC for the property "the first passenger to be collected is not the closest".

A second application of BMC is based on gathering a large number of traces. We construct a small human-readable model that explains the decision procedure of the wizard. We note that while the magic book is already a small model that approximates the wizard, its size is way too large for a human to reason about. For us, a small model is one decision tree of depth at most 4. Moreover, the magic book is a "local" function, its type is from states to actions, whereas a human is typically interested in "global" behavior, e.g., which action to take next as opposed to which passenger is collected next, respectively.

We rely on the user to supply specifications $\Omega_1, \ldots, \Omega_m$. We gather a dataset that consists of pairs of the form $(\bar{s}, i)$, for each $1 \leq i \leq m$, where $\bar{s}$ is such that when the plant starts at configuration $\bar{s}$ under the control of the wizard, then Ω_i is satisfied. To find many traces that satisfy Ω_i, we iteratively call an SMT solver. Suppose it finds a trace $\eta \in \Omega_i$. Then, before the next call, we add the constraint $\neg\eta$ to the SMT program so that η is not found again. In practice, the amortized running time of this simple algorithm is low. One reason is that generating the SMT program takes considerable time, even when comparing to the time it takes to solve it. This running time is essentially amortized over all executions since the running time of adding a single constraint is negligible. In addition, the SMT solver learns the structure of the SMT program and uses it to speed up subsequent executions.

Example 5. *Suppose we are interested in understanding if and how the wizard prioritizes collecting passengers. We consider the specifications "Passenger j is collected first", for $1 \leq j \leq k$. It can be formalized using the following constraints. The constraint $\bigwedge_{1 \leq i \leq \ell}(x_j^i = x_j^0 \wedge y_j^i = y_j^0)$ means that Passenger j is not collected since it stays in place throughout the whole trace, and we add such a constraint for all but one passenger. The constraint $\neg(x_j^\ell = x_j^0 \wedge y_j^\ell = y_j^0)$ means that Passenger j must have been collected at least once since its final position differs from his initial position. In Fig. 4 we depict a tree that we extract using these specifications.*

V. Experiments

a) Setup: We illustrate our approach using an implementation of the case study that is our running example: a taxi traveling on a grid and collecting passengers. We set the size of the grid to be $n = 10$ and the number of passengers to $m = 3$, thus the state space is almost 10^8. All simulations were programmed in Python and run on a personal computer with an Intel Core i3-4130 3.40GHz CPU, 7.7 GiB memory runnning Ubuntu.

b) Training a wizard using deep RL: The plant state in our training is a 6-tuple that, for each passenger, contains the distances to the taxi on both axes. When the taxi collects a passenger, the agent receives a reward of 100. Multi-objective RL is notoriously difficult because the agent gets confused by the various targets. We thus found it useful to add a "hint" when the taxi does not collect a passenger: at time $t > 1$, if a passenger is not collected, the agent receives a reward of $\max_{i=1,2,3}\left(1/d_{t+1,i} - 1/d_{t,i}\right)$, where $d_{j,i}$, for $j \geq 1$ and $i \in \{1, 2, 3\}$, is the manhattan distance between the taxi and passenger i at time j. We use the Python library Keras [36] and the "Adam" optimizer [37] to minimize mean squared error loss. We train a NN with two hidden layers that use a ReLU activation function and with 200 and 100 neurons, respectively, and a linear output layer. Each episode consists of 1000 steps and we train for 2000 episodes.

c) Extracting the magic book: We extract configuration-action pairs from 1000 episodes of the trained agent. We use Python's scikit-learn library [38] to fit one of the tree-based classification model to the obtained dataset. Table I depicts a comparison between the models and the wizard on 200 episodes. *Performance* refers to the total number of passengers collected in a simulation. It is encouraging that small forests with shallow trees (of depth not more than 10) approximate the wizard well.

d) Synthesis: Following expert advice: The specification we consider is "reach a gas station every t time steps", for some $t \in \mathbb{N}$. Our controllers exhibit performance that is not too far from the wizard: see Table I for the performance with $t = 30$ and synthesis based on different tree models (take into account that the wizard does not visit the gas station). We view this experiment as a success: we achieve our goal

Num. of collected passengers	DT(10)	RF(5,6)	xGB(100,10)	Wizard
Avg. performance	147	154	158	159
Max. performance	194	194	190	200
Synthesis avg. performance	122	96	–	–

TABLE I: Performances of the wizard compared to three classifiers: decision tree DT(depth), random forest RF(trees, depth), and extreme gradient boosting xGB(trees, depth). Each simulation was ran 10 times for an arbitrary $\bar{s}_0$ and time bound $T = 1000$.

Bound	Passenger 1		Passenger 2		Passenger 3	
	runtime	succ. ratio	runtime	succ. ratio	runtime	succ. ratio
6	0.26 s	82.8 %	0.25 s	85 %	0.25 s	81.2 %
7	0.30 s	76.9 %	0.30 s	87.2 %	0.30 s	84.2 %
8	0.37 s	85.2 %	0.36 s	89.9 %	0.37 s	88.7 %
9	0.44 s	85.1 %	0.47 s	82.2 %	0.49 s	79.7 %

TABLE II: Results for BMC with bounds $6-9$ using a forest with 5 trees of depth 10 as a magic book. The amortized running times for obtaining a trace, over 250 traces, and the ratio of traces that are witnesses for the wizard.

of synthesizing a correct controller that achieves favorable performance. We point out that since traditional synthesis does not address performance, a controller that it produces visits the gas station every t steps but does not collect any passenger.

e) Comparing with a shield-based approach: A shield-based controller [13], [14] consists of a shield that uses a wizard as a black box: given a plant state $\bar{s}$, the wizard is run to obtain $a = \textsc{Wiz}(\bar{s})$, then a is fed to the shield to obtain $a' \in A$, which is issued to the plant. We demonstrate how our synthesis procedure manages to open up the black-box wizard. In Fig. 5, we depict the result of an experiment in which we add a wall to the grid that was not present in training. Crossing a wall is inherently impossible for the shield-based controller since when the wizard suggests an action that is not allowed, the best the shield can do is choose an arbitrary substitute. Our controller, on the other hand, intuitively directs the taxi to areas in the grid where the magic book is "certain" of its actions (a notion which is convenient to define when the magic book is a forest). Since these positions are often located near passengers, the taxi manages to cross the wall.

f) BMC: Scalability and success rate: We use the standard state-of-the-art SMT solver Z3 [12] to solve BMC. In Table II, we consider the following specifications for XAI: "Passenger i is collected first and at time ℓ, even though it is not closest", where ℓ is the bound for BMC and for $i \in \{1,2,3\}$. We perform the following experiment 10 times and average the results. We run BMC to collect 250 traces. We depict the amortized running time of finding a trace, i.e., the total running time divided by 250. Recall that the traces witness the magic book. We count the number of traces out of the 250 that also witness the wizard, and depict their ratio. We find both results encouraging: finding a dataset of non-trivial witness traces of the wizard is feasible.

g) Wizard-based BMC: We implemented a BMC procedure that simulates the wizard instead of the magic book and ran it using Z3. We observe extremely poor scalability: an extremely modest SMT query to find a path of length 2 timed-out at 20min, and even when the initial state is set, the running time is 4.51min!

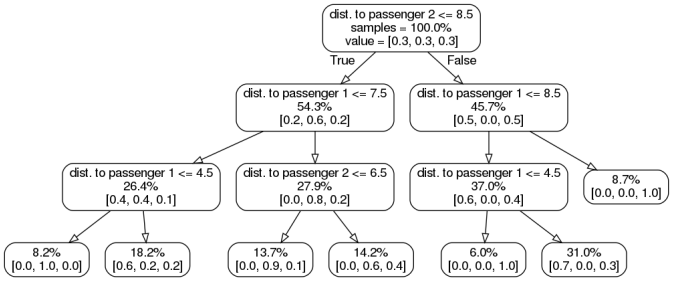

Fig. 4: A decision tree extracted from a 1200-sample dataset, obtained using BMC, of the form $(\bar{s}, i) \in S \times \{1,2,3\}$, where passenger i is collected first from the initial state $\bar{s}$.

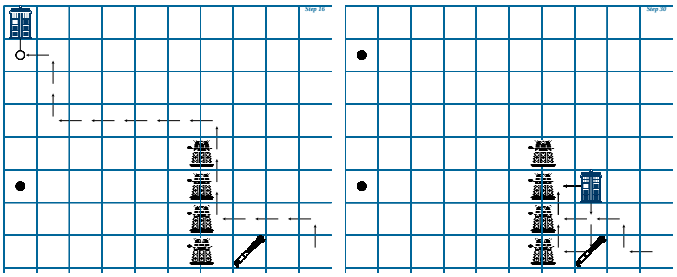

Fig. 5: Snapshots of simulations showing that a controller, synthesized using a magic-book, crosses a wall (left) whereas a shield-based controller is stuck (right).

h) BMC: Verification and Explainability: For verification, we consider the specifications "the taxi never hits the wall" and "the taxi never enters a loop in which no passenger is collected". Even though violations of these specifications were not observed in numerous simulations, we find counterexamples for both (see a depiction for the second property in Fig. 3 on the right). We illustrate explainability with the property "the closest passenger is not collected first" by depicting an example trace for it in Fig. 3 on the left. In Fig. 4, we depict a decision tree, obtained from a dataset consisting of 1200 examples, as an attempt to explain when the wizard chooses to collect passenger i first, for $i \in \{1,2,3\}$.

VI. Discussion

In this work, we address the controller-design problem using a combination of techniques from formal methods and machine learning. The challenge in this combination is that formal methods struggle with the use of neural networks (NNs). We bypass this difficulty using a novel procedure that, instead of reasoning on the NN that deep RL trains (the wizard), extracts from the wizard a small model that approximates its operation (the magic book). We illustrate the advantage of using the magic book by tackling problems that are currently out of reach for either formal methods or machine learning separately. Specifically, to the best of our knowledge, we are the first to incorporate a magic book in a reactive synthesis procedure thereby synthesizing a stand-alone controller with performance considerations. Second, we use a magic-book based BMC procedure as an XAI tool to increase the trustworthiness of the wizard.

We list several directions for future work. We find it an interesting and important problem to extract magic books with provable guarantees (see Remark 3). Another line of future work is finding other domains in which magic books can be extracted and other applications for magic books. One concrete domain is in speeding up solvers (e.g., SAT, SMT, QBF, etc). Recently, there are attempts at replacing traditional engineered heuristics with learned heuristics (e.g, [39], [40]). This approach was shown to be fruitful in [41], where an RL-based SAT solver performed less operations than a standard SAT solver. However, at runtime, the SAT solver has the upper hand since the bottleneck becomes the calls to the NN. We find it interesting to use a magic book instead of a NN in this domain so that a solver would benefit from using a learned heuristic without paying the cost of a high runtime.

Our synthesis procedure is based on an abstraction of the plant. In the future, we plan to investigate an iterative refinement scheme for the abstraction. Refinement in our setting is not standard since it includes a quantitative game (e.g., [42]), and more interesting, there is inaccuracy introduced by the magic book and wizard. Refinement can be applied both to the process of extracting the decision tree from the NN as well as improving the performance of the wizard using training.

VII. Acknowledgments

This research was supported in part by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award).

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *JMLR*, vol. 6, no. Apr, pp. 503–556, 2005.

[3] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[4] N. Frosst and G. Hinton, "Distilling a neural network into a soft decision tree," 2017.

[5] D. Shriver, D. Xu, S. Elbaum, and M. B. Dwyer, "Refactoring neural networks for verification," *arXiv preprint arXiv:1908.08026*, 2019.

[6] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE Access*, vol. 6, pp. 52 138–52 160, 2018.

[7] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proc. 16th POPL*, 1989, pp. 179–190.

[8] R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann, "Better quality in synthesis through quantitative objectives," in *Proc. 21st CAV*, 2009, pp. 140–156.

[9] A. Bohy, V. Bruyère, E. Filiot, and J. Raskin, "Synthesis from LTL specifications with mean-payoff objectives," in *Proc. 19th TACAS*, 2013, pp. 169–184.

[10] S. Almagor, O. Kupferman, J. O. Ringert, and Y. Velner, "Quantitative assume guarantee synthesis," in *Proc. 29th CAV*, 2017, pp. 353–374.

[11] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in Computers*, vol. 58, pp. 117–148, 2003.

[12] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Proc. 14th TACAS 2008*, ser. LNCS, vol. 4963. Springer, 2008, pp. 337–340. [Online]. Available: https://doi.org/10.1007/978-3-540-78800-3_24

[13] B. Könighofer, M. Alshiekh, R. Bloem, L. Humphrey, R. Könighofer, U. Topcu, and C. Wang, "Shield synthesis," *FMSD*, vol. 51, no. 2, pp. 332–361, 2017.

[14] G. Avni, R. Bloem, K. Chatterjee, T. A. Henzinger, B. Könighofer, and S. Pranger, "Run-time optimization for learned controllers through quantitative games," in *Proc. 31st CAV*, 2019, pp. 630–649.

[15] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Proc. 31st NeurIPS*, 2018, pp. 2499–2509.

[16] J. Tornblom and S. Nadjm-Tehrani, "Formal verification of input-output mappings of tree ensembles," *CoRR*, vol. abs/1905.04194, 2019, https://arxiv.org/abs/1905.04194.

[17] Y. Kazak, C. W. Barrett, G. Katz, and M. Schapira, "Verifying deep-RL-driven systems," in *Proc. of NetAI@SIGCOMM*, 2019, pp. 83–89.

[18] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Proc. 29th CAV*, 2017, pp. 97–117.

[19] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *Proc. 39th SP*, 2018, pp. 3–18.

[20] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Proc. 29th CAV*, 2017, pp. 3–29.

[21] G. Einziger, M. Goldstein, Y. Sa'ar, and I. Segall, "Verifying robustness of gradient boosted models," in *Proc. 33rd AAAI*, 2019, pp. 2446–2453.

[22] S. Drews, A. Albarghouthi, and L. D'Antoni, "Proving data-poisoning robustness in decision trees," *CoRR*, vol. abs/1912.00981, 2019. [Online]. Available: http://arxiv.org/abs/1912.00981

[23] L. Valkov, D. Chaudhari, A. Srivastava, C. A. Sutton, and S. Chaudhuri, "HOUDINI: lifelong learning as program synthesis," in *Proc. 31st NeurIPS*, 2018, pp. 8701–8712.

[24] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," in *Proc. 35th ICML*, 2018, pp. 5052–5061.

[25] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan, "An inductive synthesis framework for verifiable reinforcement learning," in *Proc. 40th PLDI*, 2019, pp. 686–701.

[26] P. Ashok, T. Brázdil, K. Chatterjee, J. Kretínský, C. H. Lampert, and V. Toman, "Strategy representation by decision trees with linear classifiers," in *Proc. 16th QEST*, ser. Lecture Notes in Computer Science, vol. 11785. Springer, 2019, pp. 109–128.

[27] P. Ashok, M. Jackermeier, P. Jagtap, J. Kretínský, M. Weininger, and M. Zamani, "dtcontrol: decision tree learning algorithms for controller representation," in *Proc. 23rd HSCC*. ACM, 2020, pp. 30:1–30:2.

[28] M. Jaeger, P. G. Jensen, K. G. Larsen, A. Legay, S. Sedwards, and J. H. Taankvist, "Teaching stratego to play ball: Optimal synthesis for continuous space MDPs," in *Proc. 17th ATVA*, 2019, pp. 81–97.

[29] J. Kretínský, G. A. Pérez, and J. Raskin, "Learning-based mean-payoff optimization in an unknown MDP under omega-regular constraints," in *Proc. 29th CONCUR*, 2018, pp. 8:1–8:18.

[30] M. Wen, R. Ehlers, and U. Topcu, "Correct-by-synthesis reinforcement learning with temporal logic constraints," in *Proc. IROS*, 2015, pp. 4983–4990.

[31] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE CSM*, vol. 12, no. 2, pp. 19–22, 1992.

[32] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[33] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD*. ACM, 2016, pp. 785–794.

[34] L. D. Pyeatt and A. E. Howe, "Decision tree function approximation in reinforcement learning," in *Proc. 3rd International Symposium on Adaptive Systems*, 2001, pp. 70–77.

[35] R. Bloem, K. Chatterjee, and B. Jobstmann, "Graph games and reactive synthesis," in *Handbook of Model Checking*, 2018, pp. 921–962.

[36] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[39] M. Soos, R. Kulkarni, and K. S. Meel, "Crystalball: Gazing in the black box of SAT solving," in *Proc. 22nd SAT*, 2019, pp. 371–387.

[40] G. Lederman, M. N. Rabe, S. Seshia, and E. A. Lee, "Learning heuristics for quantified boolean formulas through reinforcement learning," in *Proc. 8th ICLR*, 2020.
[41] E. Yolcu and B. Póczos, "Learning local search heuristics for boolean satisfiability," in *Proc. 32nd NeurIPS*, 2019, pp. 7990–8001.
[42] G. Avni and O. Kupferman, "Making weighted containment feasible: A heuristic based on simulation and abstraction," in *Proc. 23rd CONCUR*, 2012, pp. 84–99.