Verifiable RNN-Based Policies for POMDPs Under Temporal Logic Constraints

Steven Carr\textsuperscript{1}, Nils Jansen\textsuperscript{2} and Ufuk Topcu\textsuperscript{1}

\textsuperscript{1}The University of Texas at Austin
\textsuperscript{2}Radboud University, Nijmegen, The Netherlands
stevencarr@utexas.edu, n.jansen@science.ru.nl

Abstract
Recurrent neural networks (RNNs) have emerged as an effective representation of control policies in sequential decision-making problems. However, a major drawback in the application of RNN-based policies is the difficulty in providing formal guarantees on the satisfaction of behavioral specifications, e.g. safety and/or reachability. By integrating techniques from formal methods and machine learning, we propose an approach to automatically extract a finite-state controller (FSC) from an RNN, which, when composed with a finite-state system model, is amenable to existing formal verification tools. Specifically, we introduce an iterative modification to the so-called quantized bottleneck insertion technique to create an FSC as a randomized policy with memory. For the cases in which the resulting FSC fails to satisfy the specification, verification generates diagnostic information. We utilize this information to either adjust the amount of memory in the extracted FSC or perform focused retraining of the RNN. While generally applicable, we detail the resulting iterative procedure in the context of policy synthesis for partially observable Markov decision processes (POMDPs), which is known to be notoriously hard. The numerical experiments show that the proposed approach outperforms traditional POMDP synthesis methods by 3 orders of magnitude within 2\% of optimal benchmark values.

1 Introduction
Research in the reinforcement and supervised learning communities has demonstrated the utility of recurrent neural networks (RNNs) in synthesizing control policies in domains that exhibit temporal behavior [Tsoi and Back, 1997; Bakker, 2001]. The internal memory states of RNNs, such as in long short-term memory (LSTM) architectures [Hochreiter and Schmidhuber, 1997], effectively account for temporal behavior by capturing the history from sequential information [Pascaru et al., 2014]. Furthermore, in applications that suffer from incomplete information, RNNs leverage history to act as either a state or value estimator [Wierstra et al., 2007] or as a control policy [Hausknecht and Stone, 2015].

In safety-critical systems such as autonomous vehicles, policies that are guaranteed to prevent unsafe behavior are necessary. We seek to provide formal guarantees for policies represented by RNNs with respect to temporal logic [Pnueli, 1977] or reward specifications. Such a verification task is, in general, hard due to the complex, often non-linear, structures of RNNs [Mulder et al., 2015]. Existing work directly employs satisfiability-modulo-theories (SMT) [Wang et al., 2018] or mixed-integer linear program (MILP) [Akintunde et al., 2019], however, such methods not only scale exponentially in the number of variables but also rely on constructions using only rectified linear units (ReLUs).

We take an iterative and model-based approach, see Fig. 1. We extract a policy from a given RNN in the form of a finite-state controller (FSC) [Poupart and Boutilier, 2003]. First, we employ a modification of a discretization technique called quantized bottleneck insertion, introduced in [Koul et al., 2019]. Basically, the discretization facilitates a mapping of the continuous memory structure of the RNN to a pre-defined number of discrete memory states and transitions of an FSC. However, this standalone FSC without a formal model is often not sufficient to prove meaningful properties. The proposed approach relies on the exact behavior a policy induces on a specific application that can be modeled formally. We apply the extracted FSC directly to a formal model, and the
resulting restricted model is amenable for efficient verification
 techniques that certify whether a specification is satis-
 fied [Baier and Katoen, 2008].

If the specification does not hold, verification methods typ-
ically provide diagnostic information on critical parts of the
model in the form of so-called counterexamples. We propose
to utilize such counterexamples to identify improvements in
the extracted FSC or in the underlying RNN. First, increasing
the amount of memory states in the FSC may help to approx-
imate the behavior of the RNN more precisely [Koul et al.,
2019]. Second, the RNN may actually require further train-
data to induce higher-quality policies for the particular appli-
cation. Existing approaches rely, for example, on loss
visualization [Goodfellow and Vinyals, 2015], but we strive to
exploit the information we can gain from the concrete be-
havior of the RNNs with respect to a formal model. There-
fore, in order to decide whether more data are needed in
the training of the RNN or whether first the number of memory
states in the FSC should be increased, we identify those crit-
ical decisions of the current FSC that are “arbitrary”. Bas-
ically, we measure the entropy [Cover and Thomas, 2012] of
each stochastic choice over actions according to the current
FSC-based policy at critical states. That is, if the entropy is
high, the decision is deemed arbitrary despite its criticality
and further training is required.

We showcase the applicability of the proposed method on
partially observable Markov decision processes (POMDPs).
With their ability to represent sequential decision-making
problems under uncertainty and incomplete information,
these models are of particular interest in planning and con-
trol [Cassandra, 1998]. Despite their utility as a model-
ing formalism and recent algorithmic advances, policy syn-
thesis for POMDPs is hard both theoretically and practi-
cally [Meuleau et al., 1999]. For reasons outlined earlier,
RNNs have recently emerged as efficient policy representa-
tions for POMDPs [Hausknecht and Stone, 2015]. We detail
the proposed approach on POMDPs and combine the scal-
bility and flexibility of an RNN representation with the rigor
of formal verification to synthesize POMDP policies that ad-
here to temporal logic specifications.

We demonstrate the effectiveness of the proposed syn-
thesis approach on a set of POMDP benchmarks. These
benchmarks allow for a comparison to well-known POMDP
solvers, both with and without temporal logic specifications.
The numerical examples show that the proposed method (1)
is more scalable, by up to 3 orders of magnitude, than well-
known POMDP solvers and (2) achieves higher-quality re-
sults in terms of the measure of interest than other synthesis
methods that extract FSCs.

Related work. Closest to the proposed method is [Carr et
al., 2019], which introduced a verification-guided method to
train RNNs as POMDP policies. In contrast to the proposed
method, while policies are extracted from the RNNs, these
policies do not directly exhibit the memory structure of the
RNNs and are instead handcrafted based on knowledge about
the particular application.

There are three lines of related research. The first one con-
cerns the formal verification of neural network-based control
policies. Two prominent approaches [Huang et al., 2017;
Katz et al., 2017] for the class of feed-forward deep neural
networks rely on encoding neural networks as SMT problems
through adversarial examples or ReLUs architectures respec-
tively. [Akinctunde et al., 2019] concerns the direct verifica-
tion of RNNs with ReLU activation functions using SMT or
MILP. However, the scalability of these solver-based methods
suffer from the size of the input models. We circumvent this
shortcoming by our model-based approach where verification
is restricted to concrete applications followed by potential im-
provement of the RNNs.

The second relevant direction concerns the direct synthesis
of FSCs for POMDPs without neural networks. For example,
[Meuleau et al., 1999] uses a branch-and-bound method to
compute optimal FSCs, and [Junges et al., 2018] constructs
an FSC using parameter synthesis for Markov chains.

Third, existing work that concerns the extraction of FSCs
from neural networks [Zeng et al., 2019; Weiss et al., 2018;
Michalenko et al., 2019], does not integrate with formal ver-
ification to provide formal guarantees.

2 Preliminaries
A probability distribution over a set \( X \) is a function \( \mu: X \to [0, 1] \subseteq \mathbb{R} \) with \( \sum_{x \in X} \mu(x) = \mu(X) = 1 \). The set of all distributions on \( X \) is \( \text{Distr}(X) \). The support of a distribution \( \mu \) is \( \text{supp}(\mu) = \{ x \in X | \mu(x) > 0 \} \). The entropy of a distribution \( \mu \) is \( H(\mu) := -\sum_{x \in X} \mu(x) \log \mu(x) \).

**POMDPs.** A Markov decision process (MDP) is a tuple \( M = (S, A, P, \mathcal{O}) \) with a finite (or countable infinite) set \( S \) of states, a finite set \( A \) of actions, and a transition probability function \( P: S \times A \to \text{Distr}(S) \). The reward function for states and actions is given by \( r: S \times A \to \mathbb{R} \). A finite path \( \pi \) of an MDP \( M \) is a sequence of states and actions; last(\( \pi \)) is the last state of \( \pi \). The set of all finite paths is \( \text{Paths}_{\mathbb{N}}^{M} \).

**Definition 1 (POMDP).** A POMDP is a tuple \( M = (M, Z, O) \), with \( M \) the underlying MDP of \( M \), \( Z \) a finite set of observations, and \( O: S \to Z \) the observation function.

For POMDPs, observation-action sequences are based on
a finite path \( \pi \in \text{Paths}_{\mathbb{N}}^{M} \) of \( M \) and have the form: \( O(\pi) = O(s_{0}) \xrightarrow{a_{0}} O(s_{1}) \xrightarrow{a_{1}} \cdots O(s_{n}) \). The set of all finite observation-action sequences for a POMDP \( M \) is \( \text{ObsSeq}_{\mathbb{N}}^{M} \).

**Definition 2 (POMDP Policy).** An observation-based policy for a POMDP \( M \) is a function \( \gamma: \text{ObsSeq}_{\mathbb{N}}^{M} \to \text{Distr}(A) \) such that \( \text{supp}(\gamma(O(\pi))) \subseteq A \) for all \( \pi \in \text{Paths}_{\mathbb{N}}^{M} \). \( \Gamma_{Z}^{M} \) is the set of observation-based policies for \( M \).

A policy for a POMDP resolves the nondeterministic
choices in the POMDP, based on the history of previous ob-
servations, by assigning distributions over actions. A memo-
ryless observation-based policy \( \gamma \in \Gamma_{Z}^{M} \) is given by \( \gamma: Z \to \text{Distr}(A) \), i.e., decisions are based on the current observation
only. A POMDP \( M \) together with a policy \( \gamma \) yields an induced discrete-time Markov chain (MC) \( M^{\gamma} \). An MC does not contain any nondeterminism or partial observability. Our definition restricts POMDP policies to finite memory, which are typically represented as FSCs.
Definition 3 (Finite-state controller (FSC)). A $k$-FSC for a POMDP is a tuple $\mathcal{A} = (\mathcal{N}, n_1, \alpha, \delta)$ where $\mathcal{N}$ is a finite set of $k$ memory nodes, $n_1 \in \mathcal{N}$ is the initial memory node, $\alpha$ is the action mapping $\alpha : \mathcal{N} \times \mathcal{Z} \to \text{Distr}(\text{Act})$ and $\delta$ is the memory update $\delta : \mathcal{N} \times \mathcal{Z} \times \text{Act} \to \mathcal{N}$.

An FSC has the observations $\mathcal{Z}$ as input and the actions $\text{Act}$ as output. Upon observation, depending on the current memory node the FSC is in, the action mapping $\alpha$ returns a distribution over $\text{Act}$ followed by a change of memory nodes according to $\delta$. FSCs are an extension of so-called Moore machines, where the action mapping is deterministic, that is, $\alpha : \mathcal{N} \times \mathcal{Z} \to \text{Act}$, and the memory update $\delta : \mathcal{N} \times \mathcal{Z} \to \mathcal{N}$ does not depend on the choice of action.

Definition 4 (Specifications). We consider linear-time temporal logic (LTL) properties [Pnueli, 1977]. For a set of atomic propositions $\mathcal{AP}$, which are either satisfied or violated by a state, and $a \in \mathcal{AP}$, the set of LTL formulas is:

$$\Psi := a \mid (\Psi \land \Psi) \mid \neg \Psi \mid \bigcirc \Psi \lor \square \Psi \mid (\Psi \cup \Psi).$$

Intuitively, a path $\pi$ satisfies the proposition $a$ if its first state does; $(\psi_1 \land \psi_2)$ is satisfied, if $\pi$ satisfies both $\psi_1$ and $\psi_2$; $\neg \psi$ is true on $\pi$ if $\psi$ is not satisfied. The formula $\bigcirc \psi$ holds on $\pi$ if the subpath starting at the second state of $\pi$ satisfies $\psi$; $\pi$ satisfies $\square \psi$ if all suffixes of $\pi$ satisfy $\psi$. Finally, $\pi$ satisfies $(\psi_1 \cup \psi_2)$ if there is a suffix of $\pi$ that satisfies $\psi_2$ and all longer suffixes satisfy $\psi_1$. $\bigcirc \psi$ abbreviates $(\true \cup \psi)$.

For POMDPs, one wants to synthesize a policy such that the probability of satisfying an LTL-property respects a given bound, denoted $\psi = E_{\lambda}(\psi)$ for $\lambda \in \{<, \leq, \geq, >\}$ and $\lambda \in [0, 1]$. In addition, undischorded expected reward properties $\psi = E_{\lambda}(\square \psi)$ require that the expected accumulated cost until reaching a state satisfying $\psi$ respects $\lambda \in [0, 1]$. A specification $\psi$ is satisfied for POMDP $\mathcal{M}$ and $\gamma$ if it is satisfied in the MC $\mathcal{M}^\gamma$. We now define a general notion of an RNN that represents a POMDP policy.

Definition 5 (Policy network). A policy network for a POMDP is a function $\gamma : \text{ObsSeq}_{\mathcal{FS}} \to \text{Distr}(\text{Act})$.

The underlying RNN which receives sequential input in the form of (finite) observation sequences from $\text{ObsSeq}_{\mathcal{FS}}$, the output is a distribution over actions, see Fig. 4a. To be more precise, we identify the main components of such a network.

Definition 6 (Components of a policy network). A policy network $\gamma$ is sufficiently described by a hidden-state update function $\delta : \mathcal{R} \times \mathcal{Z} \times \text{Act} \to \mathcal{R}$ and an action mapping $\gamma : \mathcal{R} \to \text{Distr}(\text{Act})$.

Consider the following observation sequence:

$$O(\pi) = O(s_0) \xrightarrow{a_0} O(s_1) \xrightarrow{a_1} \cdots O(s_i)$$

The policy network receives an observation and returns an action choice. Throughout the execution of the sequence, the RNN holds a continuous hidden state $h \in \mathcal{R}$, occasionally described as an internal memory state, which captures previous information. On each transition, this hidden state is updated to include the information of the current state and the last action taken under the hidden state transition function $\delta$. From the prior observation sequence in (1), the corresponding hidden state sequence would be defined as:

$$\delta(\pi) = h_0 \xrightarrow{a_0, O(s_1)} h_1 \xrightarrow{a_1, O(s_2)} \cdots h_i$$

Additionally, the output of the policy network is expressed by the action-distribution function $\gamma(h)$, which maps the value of hidden state to a distribution over the actions. At internal memory states $h_i$, we have $\delta(h_i, O(s_i), a_i) = h_{i+1}$ and $\gamma(h_{i+1}) = \mu(\text{Act})$ for state $s_i$ on path $\pi$. Note that a policy network characterizes a well-defined POMDP policy.

3 Problem Statement

We attempt to solve two separate but related problems: (1) For a POMDP $\mathcal{M}$, a policy network $\gamma$ and a specification $\phi$, the problem is to extract an FSC $\mathcal{A}_\phi \in \Gamma^{\mathcal{M}}_2$ such that $\mathcal{M}^{\mathcal{A}_\phi} \models \phi$. (2) If the extraction process fails to produce a suitable candidate, then we determine an improved policy network $\gamma^*$ for which we can solve (1).

3.1 Outline

Fig. 2 illustrates the workflow of the proposed approach for a given POMDP $\mathcal{M}$, policy network $\gamma$ and specification $\phi$. We summarize the individual steps below and provide the technical details in the subsequent sections.

FSC Extraction. We first quantize the memory nodes of the policy network $\gamma$, that is, we discretize the memory update of the continuous memory state $h$. From this discrete representation of the memory update, we construct an FSC $\mathcal{A}_\phi \in \Gamma^{\mathcal{M}}_2$. The procedure has as input the number $B_h$ of neurons which defines a bound on the number of memory nodes in the FSC.
Verification. We use the FSC $A_e$ to resolve partial information and nondeterministic choices in the POMDP $M$, resulting in an induced MC $M^{A_e}$. We evaluate whether the given specification $\varphi$ is satisfied for this induced MC using a formal verification technique called model checking [Baier and Katoen, 2008]. If the specification $\varphi$ holds, then the synthesis is complete with output policy $A_\hat{\varphi}$. However, if $\varphi$ does not hold, then we decide if we shall increase the bound $B_h$ on the number of memory nodes or if the network needs retraining. In particular, we examine whether or not the entropy over the FSC’s action distribution is above a prescribed threshold.

Policy improvement. In the high entropy case, we increase the discretization level, that is, we increase $B_h$, and construct the FSC $A_\hat{\varphi}$ with additional memory states at its disposal. Whereas in the other case, additional memory nodes may cause the extracted FSC to be drawn from extrapolated information and we instead seek to improve the policy network. For that, we use diagnostic information in the form of counterexamples to generate new data [Carr et al., 2019].

Example 1. We consider the POMDP in Fig. 3 as a motivating example for the necessity of memory-based FSCs. The POMDP has three observations (“blue”, $s_3$ and $s_4$) where observation “blue” is received upon visiting $s_0$, $s_1$, and $s_2$. That is, the agent is unable to distinguish between these states. The specification is $\varphi = \Pr_{r \geq 0.9}(\diamond s_3)$, so the agent is to reach state $s_3$ with at least probability 0.9. In a 1-FSC (i.e. one memory node 0), we can describe an FSC $A_1$ by:

\[
\alpha(0, \text{blue}) = \begin{cases} 
\text{up} & \text{with probability } p, \\
\text{down} & \text{with probability } 1 - p,
\end{cases}
\]

\[
\delta(0, z, a) = 0 \quad \forall z \in Z, a \in \text{Act}.
\]

A 2-FSC with two memory nodes (0 and 1), see Fig. 3c, allows for greater expressivity, i.e. the policy can base its decision on larger observation sequences. With this memory structure, we can create an FSC $A_2$ that ensures the satisfaction of $\varphi$:

\[
\alpha(0, \text{blue}) = \begin{cases} 
\text{up} & \text{with probability } 1, \\
\text{down} & \text{with probability } 0,
\end{cases}
\]

\[
\alpha(1, \text{blue}) = \begin{cases} 
\text{up} & \text{with probability } 0, \\
\text{down} & \text{with probability } 1,
\end{cases}
\]

\[
\delta(0, \text{blue}, \text{up}) = 1, \\
\delta(1, \text{blue}, \text{down}) = 0.
\]

4 Policy Extraction

In this section we describe how we adapt the method called quantized bottleneck insertion [Koul et al., 2019] to extract an FSC from a given RNN. Let us first explain the relationship between the main components of a policy network $\hat{\gamma}$ (Definition 6) and an FSC $A$ (Definition 3). In particular, the hidden-state update function $\delta: \mathbb{R} \times Z \times \text{Act} \to \mathbb{R}$ takes as input a real-valued hidden state of the policy network, while the memory update function of an FSC takes a memory node from the finite set $N$. The key for linking the two is therefore a mechanism that encodes the continuous hidden state $h$ into a set $N$ of discrete memory nodes.

Policy network modification. To obtain the above linkage, we leverage an autocoder [Goodfellow et al., 2016] in the form of a quantized bottleneck network (quantized bottleneck network (QBN)) [Koul et al., 2019]. This QBN, consisting of an encoder and a decoder, is inserted into the policy network directly before the softmax layer, see Fig. 4b. In the encoder, the continuous hidden state value $h \in \mathbb{R}$ is mapped to an intermediate real-valued vector $\mathbb{R}^{B_h}$ of pre-allocated size $B_h$. The decoder then maps this intermediate vector into a discrete vector space defined by $\{0, 1, 2\}^{B_h}$. This process, illustrated in Fig. 4b, provides a mapping of the continuous hidden state $h$ into a set of possible discrete values. We denote the discrete state for $h$ by $h$ and the set of all such discrete states by $\hat{H}$. Note, that $|\hat{H}| \leq 2^{B_h}$ since not all values of the hidden state may be reached in an observation sequence. [Koul et al., 2019] has another QBN for a continuous observation space, however, we focus on discrete observations and can neglect the additional autocoder.

FSC construction. After the QBN insertion we simulate a series of executions, querying the modified RNN for action choices, on the concrete application, e.g. using a POMDP model. We form a dataset of consecutive pairs $(h_t, h_{t+1})$ of discrete states, the action $a_t$ and the observation $z_{t+1}$ that led to the transition $\{h_t, a_t, z_{t+1}, h_{t+1}\}$ at each time $t$ during the execution of the policy network. The number of accessed memory nodes $N \subseteq \hat{H}$ corresponds to the number of different discrete states $h \in \hat{H}$ in this dataset. The deterministic memory update rule $\delta(n_t, a_t, z_{t+1}) = n_{t+1}$ is obtained by constructing a $N \times (|Z| \times |\text{Act}|)$ transaction table, for a detailed description see [Koul et al., 2019]. We can additionally construct the action mapping $\alpha: N \times Z \rightarrow \text{Distr}(\text{Act})$ with $\alpha(n_t, z_t) = \mu \in \text{Distr}(\text{Act})$ by querying the softmax-output layer (see Fig. 4a) for each memory state and observation.
Improving the policy network. Our goal is to determine whether a policy network requires more training data or not. Existing approaches in supervised learning methods leverage benchmark comparisons between a train-test set using a loss function [Baum and Wilczek, 1987]. Loss visualization, proposed by [Goodfellow and Vinyals, 2015] provides a set of analytical tools to show model convergence. However, such approaches aim at continuous functions instead of the discrete representations we seek. More importantly, we leverage the information gained from a model-based approach.

Counterexamples. We first determine a set of states that are critical for satisfaction of the specification under the current policy. Consider a sequence of memory nodes and observations \((n_0, z_0) \rightarrow a_0 \cdots \rightarrow a_{t-1} \rightarrow (n_t, z_t)\) from the POMDP \(\mathcal{M}\) under the FSC \(\mathcal{A}_3\). For each of these sequences, we collect the states \(s \in S\) underlying the observations, e.g., \(O(s) = z_i\) for \(0 \leq i \leq t\). As we know the probability or expected reward for these states to satisfy the specification from previous model checking, we can now directly assess their criticality regarding the specification. We collect all pairs of memory nodes and states from \(N \times S\) that contain critical states and build the set \(\text{Crit}_{\mathcal{A}_3} \subseteq N \times S\) that serves us as a counterexample. These pairs carry the joint information of critical states and memory nodes from the policy applied to the MC and may be formalized using a so-called product construction.

Entropy measure. The average entropy across the distributions over actions at the choices induced by the counterexample set \(\text{Crit}_{\mathcal{A}_3}\) is our measure of choice to determine the level of training for the policy network. Specifically, for each pair \((n, s) \in \text{Crit}_{\mathcal{A}_3}\), we collect the distribution \(\mu \in \text{Distr}(\text{Act})\) over actions that \(s\) returns for the observation \(O(s)\) when it is in memory node \(n\). Then, we define the evaluation function \(H\) using the entropy \(\mathcal{H}(\mu)\) of the distribution \(\mu\):

\[
H : \text{Crit}_{\mathcal{A}_3} \rightarrow [0, 1] \text{ with } H(n, s) = \mathcal{H}(\mu)
\]

For high values of \(H\), the distribution is uniform across all actions and the associated policy network is likely extrapolating from unseen inputs.

In Fig. 5, we observe that when there are fewer samples and higher discretization, the extracted FSC tends to perform over actions that \(s\) when it is in memory node \(n\). Then, we define the evaluation function \(H\) using the entropy \(\mathcal{H}(\mu)\) of the distribution \(\mu\):

\[
H : \text{Crit}_{\mathcal{A}_3} \rightarrow [0, 1] \text{ with } H(n, s) = \mathcal{H}(\mu)
\]

For high values of \(H\), the distribution is uniform across all actions and the associated policy network is likely extrapolating from unseen inputs.

In Fig. 5, we observe that when there are fewer samples and higher discretization, the extracted FSC tends to perform arbitrarily. We lift the function \(H\) to the full set \(\text{Crit}_{\mathcal{A}_3}\):

\[
H_{\text{Crit}_{\mathcal{A}_3}} = \frac{1}{|\text{Crit}_{\mathcal{A}_3}|} \sum_{(n,s) \in \text{Crit}_{\mathcal{A}_3}} H(n,s)
\]  

We compare the average entropy over all components of the counterexample against a threshold \(\eta \in [0, 1]\), that is, if \(H_{\text{Crit}_{\mathcal{A}_3}} > \eta\), we will provide more training data. Vice versa, if \(H_{\text{Crit}_{\mathcal{A}_3}} \geq \eta\), we will increase the upper bound on the number of memory states in the FSC.

Example 1 (cont.). Under the working example, the policy \(\mathcal{A}_1\) was the 1-FSC with \(p = 1\) (Fig. 3b), which
<table>
<thead>
<tr>
<th>Problem</th>
<th></th>
<th></th>
<th></th>
<th>Extraction Approach</th>
<th></th>
<th>Handcrafted</th>
<th></th>
<th>PRISM-POMDP</th>
<th></th>
<th>SolvePOMDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maze(1)</td>
<td>11</td>
<td>7</td>
<td>Min</td>
<td></td>
<td></td>
<td>2</td>
<td>3.33</td>
<td>3.03</td>
<td>2.33</td>
<td>2.90</td>
</tr>
<tr>
<td>Maze(2)</td>
<td>14</td>
<td>7</td>
<td>Min</td>
<td></td>
<td></td>
<td>3</td>
<td>5.34</td>
<td>11.23</td>
<td>3.51</td>
<td>46.65</td>
</tr>
<tr>
<td>Maze(3)</td>
<td>23</td>
<td>7</td>
<td>Min</td>
<td></td>
<td></td>
<td>3</td>
<td>12.39</td>
<td>160.12</td>
<td>6</td>
<td>14.40</td>
</tr>
<tr>
<td>Maze(10)</td>
<td>38</td>
<td>7</td>
<td>Min</td>
<td></td>
<td></td>
<td>5</td>
<td>23.02</td>
<td>210.01</td>
<td>11</td>
<td>100.23</td>
</tr>
<tr>
<td>Grid(3)</td>
<td>9</td>
<td>2</td>
<td>Min</td>
<td></td>
<td></td>
<td>3</td>
<td>9.20</td>
<td>87.33</td>
<td>2</td>
<td>2.90</td>
</tr>
<tr>
<td>Grid(4)</td>
<td>16</td>
<td>2</td>
<td>Min</td>
<td></td>
<td></td>
<td>7</td>
<td>4.20</td>
<td>124.31</td>
<td>3</td>
<td>4.32</td>
</tr>
<tr>
<td>Grid(5)</td>
<td>25</td>
<td>2</td>
<td>Min</td>
<td></td>
<td></td>
<td>9</td>
<td>5.91</td>
<td>250.14</td>
<td>4</td>
<td>6.63</td>
</tr>
<tr>
<td>Grid(10)</td>
<td>100</td>
<td>2</td>
<td>Min</td>
<td></td>
<td></td>
<td>9</td>
<td>12.92</td>
<td>1031.33</td>
<td>9</td>
<td>13.63</td>
</tr>
<tr>
<td>Grid(25)</td>
<td>625</td>
<td>2</td>
<td>Min</td>
<td></td>
<td></td>
<td>16</td>
<td>35.32</td>
<td>6514.30</td>
<td>24</td>
<td>531.05</td>
</tr>
<tr>
<td>Navigation (4)</td>
<td>256</td>
<td>26</td>
<td>Max</td>
<td></td>
<td></td>
<td>8</td>
<td>0.92</td>
<td>160.32</td>
<td>8</td>
<td>0.92</td>
</tr>
<tr>
<td>Navigation (5)</td>
<td>625</td>
<td>256</td>
<td>Max</td>
<td></td>
<td></td>
<td>8</td>
<td>0.95</td>
<td>311.65</td>
<td>8</td>
<td>0.92</td>
</tr>
<tr>
<td>Navigation (10)</td>
<td>10^6</td>
<td>256</td>
<td>Max</td>
<td></td>
<td></td>
<td>8</td>
<td>0.90</td>
<td>2561.02</td>
<td>4</td>
<td>0.85</td>
</tr>
<tr>
<td>Navigation (20)</td>
<td>1.6 x 10^5</td>
<td>256</td>
<td>Max</td>
<td></td>
<td></td>
<td>9</td>
<td>0.98</td>
<td>8173.03</td>
<td>4</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 1: Synthesizing strategies for examples with expected reward and LTL specifications.

<table>
<thead>
<tr>
<th>Component</th>
<th>Time (s)</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>311.65</td>
<td>100</td>
</tr>
<tr>
<td>Training/Retraining RNN</td>
<td>205.77</td>
<td>66.0</td>
</tr>
<tr>
<td>Extracting/Applying FSC</td>
<td>80.21</td>
<td>25.7</td>
</tr>
<tr>
<td>Verification of MC</td>
<td>2.23</td>
<td>0.7</td>
</tr>
<tr>
<td>Data from countexamples/entropy check</td>
<td>7.03</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Discussion. The results are shown in Table 1. The proposed extraction approach scales to significantly larger examples than both state-of-the-art POMDP solvers which compute near-optimal policies. While the handcrafted approach scales equally well, the extraction method produces high-quality policies - within 2% of the optimum. That effect is due to our automatic extraction of suitable FSCs. Note that an optimal policy for Maze(1) can be expressed using 2 memory states. The FSC structure employed by the handcrafted method uses this structure and consequently, for the small Maze environments, the handcrafted method synthesizes higher values. Yet, with larger environments the fixed memory structure produces poor policies as more memory states are beneficial to account for the past behavior.

Acknowledgements
This work was supported by DARPA D19AP00004, ONR N00014-18-1-2829, and ARL ACC-APG-RTP W911NF.
References


