Active Reward Machine Inference From Raw State Trajectories

Mohamad Louai Shehab[1][0009−0009−7179−3881], Antoine Aspeel[2][0000−0003−3011−7122], and Necmiye Ozay[1,3][0000−0002−5552−4392]

[1] Robotics Department, University of Michigan, Ann Arbor, MI
[2] Independent Researcher
[3] Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI

Abstract. Reward machines are automaton-like structures that capture the memory required to accomplish a multi-stage task. When combined with reinforcement learning or optimal control methods, they can be used to synthesize robot policies to achieve such tasks. However, specifying a reward machine by hand, including a labeling function capturing high-level features that the decisions are based on, can be a daunting task. This paper deals with the problem of learning reward machines directly from raw state and policy information. As opposed to existing works, we assume no access to observations of rewards, labels, or machine nodes, and show what trajectory data is sufficient for learning the reward machine in this information-scarce regime. We then extend the result to an active learning setting where we incrementally query trajectory extensions to improve data (and indirectly computational) efficiency. Results are demonstrated with several grid world examples.

Keywords: Inverse Reinforcement Learning · Reward Machines · Multi-stage tasks.

1 Introduction

Multi-stage tasks are ubiquitous in robotics applications, as real-world objectives are rarely achieved through a single atomic action but instead require the coordinated execution of sequential and interdependent subtasks [1–3]. These robots have to operate under temporally extended objectives in which task success depends on satisfying intermediate goals in a specific order, motivating the use of hierarchical and modular task representations [4]. Reward machines provide a principled and expressive formalism for representing such multi-stage task structure by encoding task progress as a finite-state automaton whose transitions are triggered by high-level events or propositions observed during execution [5–9].

While Reward Machines (RMs) offer a powerful framework for structured task execution, their utility is often bottlenecked by the requirement for human experts to manually specify the underlying automaton. In complex, real-world environments, defining the exact logical transitions and propositional triggers

for a task is not only labor-intensive but also prone to specification errors that can lead to unintended robotic behaviors [10]. Consequently, there is a growing imperative to develop algorithms capable of learning RMs directly from experience [6, 11, 12]. This automated RM inference allows a robot to autonomously discover the latent logical structure of a task, identifying the "hidden" stages that define successful execution without explicit human oversight.

Existing literature on learning reward machines generally falls into three categories: those assuming ground-truth labels for states or rewards to find consistent automata [13–17], those utilizing active learning or L^* oracles to query membership and conjectures [18–20], and those combining automata synthesis with reinforcement learning in interactive environments [21–23]. While some approaches rely on observing demonstrations [7, 24], they are often restricted to single-stage tasks where the RM serves primarily for reward shaping rather than complex temporal logic. Other works learn reward machines for multi-staged tasks purely from demonstrations [12]. A critical bottleneck in these works is the reliance on a predefined labeling function that maps low-level states to high-level propositions. Recent efforts have begun addressing labeling function ambiguity by accounting for noise and uncertainty in label assignments [25–27]; however, they still assume the existence of a noisy labeling function. In contrast, our framework is the first to learn both the labeling function and the reward machine from scratch using only state-based traces, eliminating the need for any prior symbolic knowledge or predefined event detectors.

Notation: For a finite set X, we denote by $|X|$ its cardinality. The set of all probability distributions over X is denoted by $\Delta(X)$. The set of all finite sequences with elements in X is denoted by X^*. For two sets X and Y, their Cartesian product is $X \times Y$, and the Cartesian power of X of order n is denoted by X^n. The set of real numbers is denoted by $\mathbb{R}$. Logical conjunction and disjunction are denoted by $\wedge$ and $\vee$, respectively. The expectation operator is denoted by $\mathbb{E}$.

2 Preliminaries

2.1 Markov decision processes and reward machines

A *Markov Decision Process* (MDP) is a tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mu_0, \gamma, r),$$

where $\mathcal{S}$ is the finite set of states, $\mathcal{A}$ is the finite set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the Markovian transition kernel, $\mu_0 \in \Delta(\mathcal{S})$ is the initial state distribution, $\gamma \in [0, 1)$ is the discount factor, and $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function. We refer to a MDP without the reward r as an *MDP model*, and denote it by $\mathcal{M} \setminus r$.

A *Reward Machine* (RM) is a tuple

$$\mathcal{R} = (\mathcal{U}, u_I, \mathrm{AP}, \delta_{\mathbf{u}}, \delta_{\mathbf{r}}),$$

where $\mathcal{U}$ is the finite set of nodes, $u_I \in \mathcal{U}$ is the initial node, AP is the set of atomic propositions (also called input alphabet), $\delta_{\mathbf{u}} : \mathcal{U} \times \text{AP} \to \mathcal{U}$ is the (deterministic) transition function, and $\delta_{\mathbf{r}} : \mathcal{U} \times \text{AP} \to \mathbb{R}$ is the output function. A reward machine without its output function is named a *reward machine model*. We extend the definition of the transition function to define $\delta^*_{\mathbf{u}} : \mathcal{U} \times (\text{AP})^* \to \mathcal{U}$ as $\delta^*_{\mathbf{u}}(u, l_0, \cdots, l_k) = \delta_{\mathbf{u}}(\cdots(\delta_{\mathbf{u}}(\delta_{\mathbf{u}}(u, l_0), l_1), \cdots, l_k)$.

Finally, a *labeling function* (compatible with an MDP $\mathcal{M}$ and a RM $\mathcal{R}$) is a function $L : \mathcal{S} \to \text{AP}$ which associates an atomic proposition of a RM to each state of an MDP.

It is common to introduce the notion of labeled MDP as a pair formed by an MDP and a compatible labeling function. However, in the problem we are interested in, both the RM and the labeling function are unknown. Consequently, it will make our notation simpler to consider a labeled RM instead. Formally, a *labeled RM* refers to a pair composed by a reward machine and a (compatible) labeling function $\mathcal{R}_L = (\mathcal{R}, \mathcal{S}, L)$. A *labeled RM model* is a labeled RM without its output function $\delta_{\mathbf{r}}$ and is denoted by $\mathcal{G}$. In the next section, we show how a compatible labeling function allows to "connect" an MDP model with a RM.

2.2 Product MDP

An MDP model $\mathcal{M} \setminus r$ together with a reward machine $\mathcal{R}$ and a compatible labeling function L allow to define a *product MDP*

$$\mathcal{M}_{\text{Prod}} = (\mathcal{S}', \mathcal{A}', \mathcal{P}', \mu'_0, \gamma', r'),$$

where $\mathcal{S}' = \mathcal{S} \times \mathcal{U}$, $\mathcal{A}' = \mathcal{A}$, $\mathcal{P}'(s', u'|s, u, a) = \mathcal{P}(s'|s, a)\mathbf{1}(u' = \delta_{\mathbf{u}}(u, L(s')))$, $\gamma' = \gamma$, $\mu'_0 \in \Delta(\mathcal{S} \times \mathcal{U})$ with $\mu'_0(s, u) = \mu_0(s)\mathbf{1}(u = u_I)$ and $r'(s, u, a, s', u') = \delta_{\mathbf{r}}(u, L(s'))$, where $\mathbf{1}(p)$ is one if p is true and zero otherwise. To make the notation compact, we denote the product state by $\bar{s} = (s, u)$.

A *trajectory* of the product MDP $\mathcal{M}_{\text{Prod}}$ is a sequence $(\bar{s}_\emptyset, a_\emptyset, \bar{s}_0, a_0, \bar{s}_1, a_1, \cdots)$, where $\bar{s}_\emptyset = (\emptyset, u_I)$ and $a_\emptyset = \emptyset$. An initial state s_0 is sampled from μ_0. The introduction of $\bar{s}_\emptyset$ and $a_\emptyset$ at the start of the trajectory is to ensure that s_0 induces a transition in the reward machine. The reward machine thus transitions to $u_0 = \delta_{\mathbf{u}}(u_I, L(s_0))$. The agent then takes action a_0 and transitions to s_1. Similarly, the reward machine transitions to $u_1 = \delta_{\mathbf{u}}(u_0, L(s_1))$. The same procedure continues infinitely. We consider the product policy $\pi_{\text{Prod}} : \text{Dom}_{\text{Prod}} \to \Delta(\mathcal{A})$ where $\text{Dom}_{\text{Prod}} \subseteq \mathcal{S} \times \mathcal{U}$ is the set of accessible (s, u) pairs in the product MDP. This policy is a function that describes an agent's behavior by specifying an action distribution at each state. We consider the Maximum Entropy Reinforcement Learning (MaxEntRL) objective given by:

$$J_{\text{MaxEnt}}(\pi; r') = \mathbb{E}^{\pi}_{\mu_0}[\sum_{t=0}^{\infty} \gamma^t \Big(r'(\bar{s}_t, a_t, \bar{s}_{t+1}) + \lambda \mathcal{H}(\pi(.|\bar{s}_t)) \Big)], \tag{1}$$

where $\lambda > 0$ is a regularization parameter, and $\mathcal{H}(\pi(.|\bar{s})) = -\sum_{a \in \mathcal{A}} \pi(a|\bar{s}) \log(\pi(a|\bar{s}))$ is the entropy of the policy π. The expectation is with respect to the probability distribution $\mathbb{P}^{\pi}_{\mu_0}$, the induced distribution over infinite trajectories following

π, μ_0, and the Markovian transition kernel $\mathcal{P}'$ [28]. The optimal policy π^*_{Prod}, corresponding to a reward function r', is the maximizer of (1), i.e.,

$$\pi^*_{\text{Prod}} = \arg\max_{\pi} J_{\text{MaxEnt}}(\pi; r'). \tag{2}$$

As shown in [28], the maximizer is unique and this policy is well defined.

3 Problem Statement

We investigate the problem of learning a reward machine that makes a policy optimal, without assuming that the rewards, the machine nodes, or the atomic propositions are observed. This problem is ill-posed since several reward machines could make a policy optimal. To write this problem formally, we first introduce the notion of history policy (which captures the information available for solving this learning problem), and then the notion of policy equivalence (which characterizes equivalent solutions of this learning problem).

3.1 History Policy and Reward Machine Equivalence

Consider an MDP model, a RM, and a compatible labeling function. As shown in Section 2.2, this allows to define the product MDP and the corresponding optimal product policy. The *history policy*, denoted $\pi_{\text{h}} : \text{Dom}_{\text{h}} \to \Delta(\mathcal{A})$ is defined as

$$\pi_{\text{h}}(a|s,\tau) = \pi_{\text{Prod}}(a|s, \delta^\star_u(u_I, L(\tau))). \tag{3}$$

The domain $\text{Dom}_{\text{h}} \subseteq \mathcal{S} \times \mathcal{S}^*$ is the largest set for which the right-hand side of (3) is well defined. In (3), $\tau \in \mathcal{S}^*$ is a trajectory of states leading up to the current state $s \in \mathcal{S}$. While π_{Prod}, δ_u, and L are not known, the history policy π_{h} is assumed to be available[1]. In that sense, the history policy serves as a state-only representation of the product policy, capturing the agent's behavior through the sequence of observable MDP states. We say that the product policy π_{Prod} *induces* π_{h}.

The history policy can take an arbitrarily long trajectory τ as argument. In contrast, we define the *depth-l restriction* of the history policy, denoted as π^l_{h}, by restricting its domain to trajectories of length at most l, i.e., the domain of π^l_{h} is $\text{Dom}_{\text{h}} \cap \left(\mathcal{S} \times \cup_{j=1}^{l} \mathcal{S}^j\right)$.

As mentioned before, the inverse reinforcement learning problem we are interested in can have multiple solutions. This is captured by the following definition.

Definition 1. *Two labeled reward machines are **policy-equivalent** with respect to an MDP model if the optimal product policies for each of the labeled reward machines induce the same history policy. Among all the labeled reward machines that are policy equivalent with respect to an MDP model, we define a **minimal** reward machine as one with the fewest number of nodes.*

[1] While we make this assumption for simplicity, in practice, instead of π_{h}, one has access to state-action trajectories of an agent implementing π_{h}. Then, π_{h} can be estimated by a sample average. A discussion on the effects of such estimation can be found in [12].

3.2 Formal problem statement

We now have all the ingredients to formalize the problems we are interested in. For an MDP model and labeled RM, consider the induced optimal history policy. Knowing the MDP model and (a depth-l^* restriction of) the history policy, is it possible to recover a labeled RM that is policy equivalent to the true one? This research question can be divided in the following:

(P1) Does there always exist a depth l^* such that, given the MDP model $\mathcal{M}$, an upper bound $u_{\max}$ on the number of nodes of the underlying reward machine and the depth-l^* restriction π_{h}^{l} of the true history policy, it is possible to learn a labeled reward machine that is policy-equivalent to the underlying one?

(P2) If l^* in problem (P1) exists, find a minimal labeled reward machine that is policy-equivalent to the underlying one.

4 Methodology

The problem of learning a reward machine (RM) directly from policies when the labels are known has previously been addressed in [29] with a two step process: 1) a Boolean Satisfiability (SAT) problem [30, 31] to learn a RM model; 2) a structured IRL problem that uses the product of the labeled RM model from step 1 and the MDP to recover the reward function. We present in this section how to extend this framework, and in particular step 1, to the more general and challenging setting in which the labeling function is unknown and must be learned jointly with the reward machine model.

4.1 Learning a Labeled Reward Machine

The unknown quantities that we aim to learn are the RM transition function $\delta_{\mathbf{u}}$, the labeling function L, and the output function $\delta_{\mathbf{r}}$. Learning $\delta_{\mathbf{u}}$, and L corresponds to the generalization of step 1 of the process described above. Learning an output function, i.e., a reward function for the product, consistent with a policy corresponds to step 2 and has been addressed in prior works [32, 29]. Therefore, this paper focuses on inferring $\delta_{\mathbf{u}}$ and L. In this section, we present a SAT problem allowing to learn $\delta_{\mathbf{u}}$ and L, i.e., a labeled reward machine model denoted $\hat{\mathcal{G}}$. Without loss of generality, let us write $\mathcal{S} = \{1, \ldots, |\mathcal{S}|\}$, $\mathcal{U} = \{1, \ldots, |\mathcal{U}|\}$, $\mathrm{AP} = \{1, \ldots, |\mathrm{AP}|\}$, and consider $u_I = 1$ and $L(1) = 1$.

The transition function $\delta_{\mathbf{u}}$ and the labeling function L are encoded by binary values as follows:

$$b_{jpi} = \begin{cases} 1 & \text{if } \delta_{\mathbf{u}}(i,p) = j, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \quad \mathbf{L}_{pk} = \begin{cases} 1 & \text{if } L(k) = p, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $i, j = 1, \ldots, |\mathcal{U}|$, $p = 1, \ldots, |\mathrm{AP}|$, and $k = 1, \ldots, |\mathcal{S}|$. The core constraints in the SAT formulation arise from negative examples, which follow from the following lemma.

Lemma 1. *Let $\tau, \tau' \in \mathcal{S}^*$ be two state trajectories. If $\pi_{\mathrm{h}}(a|s,\tau) \neq \pi_{\mathrm{h}}(a|s,\tau')$ for some $(s,a) \in \mathcal{S} \times \mathcal{A}$, then $\delta^*_{\mathbf{u}}(u_I, L(\tau)) \neq \delta_{\mathbf{u}}(u_I, L(\tau'))$.*

Proof. It follows directly from the definition of the history policy (equation (3)).

A pair of state trajectories $\{\tau, \tau'\}$ that satisfies the assumption of Lemma 1 is called a *negative example*, meaning the atomic proposition trajectories $L(\tau)$ and $L(\tau')$ should lead to different reward machine nodes starting at u_I. The following set collects all negative examples of length at most l:

$$\mathcal{E}_l^- = \left\{ \{\tau, \tau'\} \;\middle|\; \pi^l_{\mathrm{h}}(a|s,\tau) \neq \pi^l_{\mathrm{h}}(a|s,\tau') \text{ for some } (s,a) \in \mathcal{S} \times \mathcal{A} \right\}. \tag{5}$$

Note that τ and τ' may have different lengths (both not larger than l). For each pair $\{\tau, \tau'\} \in \mathcal{E}_l^-$, Lemma 1 imposes some constraints on $\delta_{\mathbf{u}}$ and L. To write these constraints via our binary encoding we need to introduce some notation. First, let us define the binary matrices $(B_p)_{ji} = b_{jpi}$ and note that they satisfy $(B_p)_{ji} = 1$ if and only if $\delta_{\mathbf{u}}(i,p) = j$. In words, B_p represents the transitions in the RM for a given atomic proposition p. Next, for a state $k \in \mathcal{S}$, consider the matrix

$$M_k = (B_1 \wedge^\star \mathbf{L}_{1,k}) \;\vee\; \cdots \;\vee\; (B_{|\mathrm{AP}|} \wedge^\star \mathbf{L}_{|\mathrm{AP}|,k}), \tag{6}$$

where $\wedge^\star$ denotes point-wise conjunction between a Boolean matrix and a Boolean scalar. This binary matrix satisfies $(M_k)_{ji} = 1$ if and only if $\delta_{\mathbf{u}}(i, L(k)) = j$. In words, M_k represents the transitions in the RM for a given MDP state k. Finally, for a state trajectory $\tau = (k_1, \ldots, k_t) \in \mathcal{S}^t$, consider the binary vector

$$v_\tau = M_{k_t} M_{k_{t-1}} \ldots M_{k_1} \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^\top,$$

which satisfies $(v_\tau)_i = 1$ if and only if $\delta^*(u_I, L(\tau)) = i$ (let us remind that we assumed $u_I = 1$). Here, v_τ represents the node at which the RM will be after emitting the labels of the state trajectory τ. For a pair of negative examples $\{\tau, \tau'\} \in \mathcal{E}_l^-$, the condition $\delta^*_{\mathbf{u}}(u_I, L(\tau)) \neq \delta^*_{\mathbf{u}}(u_I, L(\tau'))$ given by Lemma 1 can be written $v_\tau \neq v_{\tau'}$.

Overall, the SAT problem that encodes the learning of $\delta_{\mathbf{u}}$ and L is the following:

Problem 1. For a fixed l, find b_{jpi} and $\mathbf{L}_{pk}$ for $i, j = 1, \ldots, |\mathcal{U}|$, $p = 1, \ldots, |\mathrm{AP}|$, and $k = 1, \ldots, |\mathcal{S}|$ such that:

$$\sum_j b_{jpi} = 1 \qquad (\delta_{\mathbf{u}} \text{ is a function}) \tag{7a}$$

$$\sum_p \mathbf{L}_{pk} = 1 \qquad (L \text{ is a function}) \tag{7b}$$

$$\mathbf{L}_{1,1} = 1 \qquad (\text{Anchoring: } L(1) = 1) \tag{7c}$$

$$\forall \{\tau, \tau'\} \in \mathcal{E}_l^- : \; v_\tau \neq v_{\tau'} \qquad (\text{Compatibility with negative examples}) \tag{7d}$$

$$b_{jpi} = 1 \Rightarrow b_{jpj} = 1 \qquad (\text{Non-stuttering, optional constraint}) \tag{7e}$$

where each constraint must hold for all free indices.

Constraints (7a) and (7b) ensure that $\delta_{\mathbf{u}}$ and L are well defined, respectively. Constraint (7c) removes some solutions which are equivalent up to permutation. Constraint (7d) enforces the conditions given by Lemma 1. Finally, constraint (7e) allows to enforce some prior knowledge on the reward machine, when such information exits. More precisely, it enforces

$$\forall i, j, p : \delta_{\mathbf{u}}(i, p) = j \Rightarrow \delta_{\mathbf{u}}(j, p) = j$$

which holds if the underlying task is multi-stage and duration-insensitive (i.e., stutter-invariant). This constraint prevents repeated self-transitions under the same proposition and enables trace compression [15, 12]. Note that in Problem 1, it is enough to know an upper bound $u_{\max}$ on $|\mathcal{U}|$. The same is true for $|\mathrm{AP}|$, but one can always choose $|\mathrm{AP}| = |\mathcal{S}|$. Indeed, if there are more atomic propositions than states, one can consider only the atomic propositions in $L(\mathcal{S})$ whose cardinality is at most $|\mathcal{S}|$. Next, we present the main theoretical result of this section. Proposition 1 below specifies the required l^* from Section 3.2.

Proposition 1: Sufficient Depth

Given an MDP model, an upper bound $u_{\max}$ on the number of nodes of the underlying reward machine and the depth-l^* restriction $\pi_{\mathrm{h}}^{l^*}$ of some history policy π_{h}, where $l^* = |\mathcal{S}|u_{\max}^2$, Problem 1 is satisfiable with $l = l^*$ if and only if it is satisfiable for all $l > l^*$.

4.2 Proof of Proposition 1

In order to prove Proposition 1, let us first introduce the notion of synchronized labeled reward machine model. It allows to run two labeled reward machine models in parallel. It will be used to compare the ground truth labeled reward machine with the learned one.

Definition 2. *Let* $\mathcal{G}_1 = (\mathcal{U}_1, u_I^1, \mathrm{AP}^1, \delta_{\mathbf{u}}^1, \mathcal{S}, L^1)$, $\mathcal{G}_2 = (\mathcal{U}_2, u_I^2, \mathrm{AP}^2, \delta_{\mathbf{u}}^2, \mathcal{S}, L^2)$ *be two labeled reward machine models with the domains of* L^1 *and* L^2 *being a same set* $\mathcal{S}$*. The* ***synchronized labeled reward machine model*** *is the labeled reward machine model defined as follows:*

$$\begin{aligned}
\mathcal{G}^{\mathrm{sync}} &= (\mathcal{U}^{\mathrm{sync}}, u_I^{\mathrm{sync}}, \mathrm{AP}^{\mathrm{sync}}, \delta_{\mathbf{u}}^{\mathrm{sync}}, \mathcal{S}, L^{\mathrm{sync}}) \\
\mathcal{U}^{\mathrm{sync}} &= \mathcal{U}_1 \times \mathcal{U}_2, \\
u_I^{\mathrm{sync}} &= (u_I^1, u_I^2), \\
\mathrm{AP}^{\mathrm{sync}} &= \mathrm{AP}^1 \times \mathrm{AP}^2 \\
\delta_{\mathbf{u}}^{\mathrm{sync}}((u^1, u^2), (l^1, l^2)) &= (\delta_{\mathbf{u}}^1(u^1, l^1), \delta_{\mathbf{u}}^2(u^2, l^2)) \\
L^{\mathrm{sync}}(s) &= (L^1(s), L^2(s)).
\end{aligned}$$

The following definition introduces cycles in a product MDP. The core of the proof relies on removing cycles in the synchronized product MDP model.

Definition 3. *Let $\mathcal{M} \setminus r$ be an MDP model and $\mathcal{G}$ be a (compatible) labeled reward machine model. Let $\mathcal{M}_{\text{Prod}}$ be the corresponding product MDP model. Given a state trajectory $\tau = (s_1, s_2, \cdots, s_t) \in \mathcal{S}^*$, we say that a subsequence $s_{i:j}$ of τ is a **cycle** in $\mathcal{M}_{\text{Prod}}$ if $s_i = s_j$ and $\delta^*_{\boldsymbol{u}}(u_I, L(s_{:i})) = \delta^*_{\boldsymbol{u}}(u_I, L(s_{:j}))$.*

Proof (of Proposition 1).

We will refer to Problem 1 with a given l by SAT_l. Let $j > l^*$ be a natural number.

First, let us prove the "if" direction. Assume that SAT_j has a solution. Then, it satisfies constraint (7d) for all $\{\tau, \tau'\} \in \mathcal{E}_j^-$. But since $l^* < j$, $\mathcal{E}_{l^*}^- \subseteq \mathcal{E}_j^-$. Since removing constraints can not make a solution infeasible, the solution of SAT_j is also a solution of SAT_{l^*}.

Second, let us prove the "only if" direction. By contradiction, assume that SAT_{l^*} has a solution which is not a solution to SAT_j. This solution defines functions $\hat{\delta}_{\mathbf{u}}$ and $\hat{L}$ through the binary encoding (4). Since this is not a solution to SAT_j, there exists a pair of negative examples $\{\tau, \tau'\} \in \mathcal{E}_j^-$ which does not satisfy condition (7d). That is, τ, τ' satisfy

$$\pi_{\text{h}}(a|s, \tau) \neq \pi_{\text{h}}(a|s, \tau') \tag{8}$$

for some $(s, a) \in \mathcal{S} \times \mathcal{A}$, and

$$\hat{\delta}_{\mathbf{u}}(u_I, \hat{L}(\tau)) = \hat{\delta}_{\mathbf{u}}(u_I, \hat{L}(\tau')). \tag{9}$$

Denote by $\mathcal{G}^{\text{sync}}$ the synchronized product between the true labeled reward machine model $\mathcal{G} = (\mathcal{U}, u_I, \text{AP}, \delta_{\mathbf{u}}, \mathcal{S}, L)$ and the learned one $\hat{\mathcal{G}}$. Consider the product MDP model $\mathcal{M}_{prod,m}$ obtained from the MDP model and the synchronized labeled RM model. A state in $\mathcal{M}_{prod,m}$ is a tuple $(s, u, \hat{u}) \in \mathcal{S} \times \mathcal{U} \times \hat{\mathcal{U}}$, which shows that $\mathcal{M}_{prod,m}$ contains $|\mathcal{S}||\mathcal{U}||\hat{\mathcal{U}}| \leq |\mathcal{S}|u^2_{\max} = l^*$ states.

Now, consider the trajectory $\bar{\tau}$ (resp. $\bar{\tau}'$) obtained by removing cycles of τ (resp. τ') in $\mathcal{M}_{prod,m}$. Since $\mathcal{M}_{prod,m}$ has at most l^* states, this can be repeated until $|\bar{\tau}| \leq l^*$ (resp. $|\bar{\tau}'| \leq l^*$). Since only cycles have been removed, the corresponding nodes in the synchronized labeled RM model stay unchanged, i.e.,

$$\delta^{\text{sync},*}_{\mathbf{u}}(u^{\text{sync}}_I, L^{\text{sync}}(\tau)) = \delta^{\text{sync},*}_{\mathbf{u}}(u^{\text{sync}}_I, L^{\text{sync}}(\bar{\tau})),$$

and similarly for τ'. By definition of the synchronized labeled RM model, this gives

$$\delta^*_{\mathbf{u}}(u_I, L(\tau)) = \delta^*_{\mathbf{u}}(u_I, L(\bar{\tau})), \tag{10}$$

$$\hat{\delta}^*_{\mathbf{u}}(u_I, \hat{L}(\tau)) = \hat{\delta}^*_{\mathbf{u}}(u_I, \hat{L}(\bar{\tau})), \tag{11}$$

and similarly for τ'.

It follows from (10) and (3) that $\pi_{\text{h}}(a|s, \tau) = \pi_{\text{h}}(a|s, \bar{\tau})$, and similarly for τ'. Consequently, (8) implies $\pi_{\text{h}}(a|s, \bar{\tau}) \neq \pi_{\text{h}}(a|s, \bar{\tau}')$. That is, the pair $\{\bar{\tau}, \bar{\tau}'\}$ is a negative example, i.e., $\{\bar{\tau}, \bar{\tau}'\} \in \mathcal{E}_{l^*}^-$. In addition, it follows from (11) and (9) that $\hat{\delta}_{\mathbf{u}}(u_I, \hat{L}(\bar{\tau})) = \hat{\delta}_{\mathbf{u}}(u_I, \hat{L}(\bar{\tau}'))$. Overall, we have shown that the pair $\{\bar{\tau}, \bar{\tau}'\} \in \mathcal{E}_{l^*}^-$ contradicts Lemma 1 which is encoded as constraint (7d). Therefore, the solution to SAT_{l^*} does not satisfy constraint (7d), a contradiction.

4.3 Active Extension of the History Policy

One major bottleneck for solving Problem 1 comes from encoding all the negative examples in (7d) given a depth-l restriction of the history policy. Since the number of state trajectories for a standard stochastic MDP grows exponentially in the depth, representing (or storing) the history policy becomes increasingly infeasible, even for moderate depths and small state spaces. However, our key observation is that exhausting all the possible paths of the history policy is not necessary to shrink the solution set. We formalize this observation as follows.

Observation 1: Not All Trajectories are Created Equal

Suppose that we solved the SAT problem with a depth-l restriction of the history policy, which is less than the sufficient depth l^*, and obtained a solution set of candidate labeled reward machine models. Let τ be a length $(l + k)$ state trajectory, for any $k \geq 1$, and let $\tau_{:l}$ be the first l states in τ. Finally, let $\delta_{\mathbf{u}}, L$ be a ground truth transition function and labeling function respectively. If $\delta^*_{\mathbf{u}}(u_I, L(\tau)) = \delta^*_{\mathbf{u}}(u_I, L(\tau_{:l}))$, then τ will not reduce the candidate solution set, as it cannot introduce any new negative examples.

While we do not know $\delta_{\mathbf{u}}$ or L, the above observation aims at highlighting that many paths in the history policy are redundant when it comes to shrinking the candidate solution set. Hence, this motivates designing an active extension algorithm, which reduces the memory and computation requirements of fully extending the history policy. Our strategy adopts a *volume-removal* approach to active learning, where queries are selected to significantly reduce the number of hypotheses consistent with current observations. Similar volume-reduction techniques have proven effective in preference-based reward learning [33–36].

In our setting, we query trajectory extensions (histories) that are expected to most rapidly eliminate candidate labeled reward machine models consistent with the current depth history policy. For a given depth $l < l^*$, let the set of feasible solutions of Problem 1 be denoted $\mathcal{P}_{\text{feasible}} = \{\hat{\mathcal{G}}_i\}_{i=1}^N$, where N is the total number of feasible solutions. This depth l represents the *burn-in* cost in order to obtain a reasonably sized solution set. The key idea is to search for state trajectory pairs $\{\tau, \tau'\}$ for which half of the labeled reward machine models in the solution set end up in the same node, and the other half does not. If any pair $\{\tau, \tau'\}$ turns out to be an actual negative example when querying the extended ground truth history policy, then we have eliminated half of the reward machines in the solution set by just adding a single negative example. To formalize this, let $\mathfrak{B}$ be the query budget, which represents the maximum number of state trajectory pairs that we can query the history policy by. Our active learning algorithm runs as follows:

Algorithm 1: Active Extension Algorithm

1. **Initialize:** Start with an empty candidate set $\mathcal{C} = \emptyset$.
2. **Subsample:** Sample a subset of the feasible SAT solutions $\mathcal{P}_{\text{active}} \triangleq \{\hat{\mathcal{G}}_i\}_{i=1}^{N_{\text{active}}}$.
3. **Generate Candidates:** For each $\hat{\mathcal{G}}_i = (\hat{\mathcal{U}}, \hat{u}_I, \text{AP}^i, \hat{\delta}^i_{\mathbf{u}}, \mathcal{S}, \hat{L}^i) \in \mathcal{P}_{\text{active}}$, sample a random target node $u_{\text{target}} \in \hat{\mathcal{U}}$ and use randomized DFS [37] to find trajectory pairs $\{\tau, \tau'\}$ of length $l+1$ ending in the same MDP state such that $\hat{\delta}^{i,*}_{\mathbf{u}}(\hat{u}_I, \hat{L}^i(\tau)) = \hat{\delta}^{i,*}_{\mathbf{u}}(\hat{u}_I, \hat{L}^i(\tau')) = u_{\text{target}}$. Add these pairs to $\mathcal{C}$.
4. **Evaluate Quality:** For each $\{\tau, \tau'\} \in \mathcal{C}$, for each labeled reward machine model $\hat{\mathcal{G}}_i \in \mathcal{P}_{\text{active}}$ with transition function $\hat{\delta}^i_{\mathbf{u}}$ and labeling function $\hat{L}^i$, let $u_i = \hat{\delta}^i_{\mathbf{u}}(u_I, \hat{L}^i(\tau)), u'_i = \hat{\delta}^i_{\mathbf{u}}(u_I, \hat{L}^i(\tau'))$, and define the *quality* of $\{\tau, \tau'\}$ to be:

$$\texttt{quality}(\tau, \tau') = \min \left\{ \sum_{i=1}^{N_{\text{active}}} \mathbf{1}(u_i = u'_i), \sum_{i=1}^{N_{\text{active}}} \mathbf{1}(u_i \neq u'_i) \right\}, \quad (12)$$

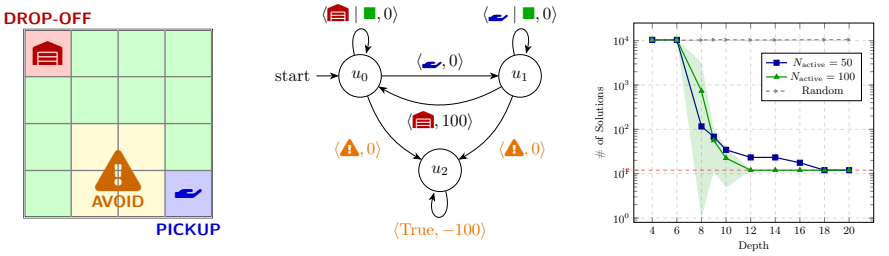
   which counts how many models in $\mathcal{P}_{\text{active}}$ predict a node collapse ($u_i = u'_i$) versus a node separation ($u_i \neq u'_i$) when traversed with $\{\tau, \tau'\}$.
5. **Query:** Sort the trajectory pairs based on the quality metric and query the history policy for the top $\mathfrak{B}$ pairs. If $\exists s^*, a^*$ such that $\pi_{\text{h}}(a^*|s^*, \tau) \neq \pi_{\text{h}}(a^*|s^*, \tau')$, add $\{\tau, \tau'\}$ to the set of negative examples.
6. **Refine:** Resolve the SAT problem incrementally with the new negative examples and update the feasible solution set $\mathcal{P}_{\text{feasible}}$.
7. **Iterate/Terminate:** If the feasible solution set has converged to a single model up-to-renaming, terminate; otherwise, increment $l = l+1$ and return to Step 2.

We note that the quality measure for a trajectory pair, $\texttt{quality}(\tau, \tau')$, is maximized when the pair bisects the candidate solution set. Also, given that exhaustive enumeration of all possible paths in Step 3 is computationally prohibitive, we employ a randomized Depth First Search (DFS) algorithm [37] with an upper bound on the size of $\mathcal{C}$[2]. This allows the exploration of deeper paths within the history policy tree than exhaustive search allows. Another algorithmic optimization we employ is re-solving the SAT problem incrementally by simply adding the newly discovered negative examples to the existing SAT instance. This allows us to increment the depth in Step 7 of the algorithm without re-solving with the exhaustive history policy (essentially the starting burn-in depth is fixed). We evaluate the empirical performance of this active approach in the experiments section.

[2] In our experiments, we set $|\mathcal{C}| \leq 10,000$.

5 Experiments

To evaluate the effectiveness of our proposed framework, we consider a grid world navigation environment (Figure 2a). Each cell in this 4×4 grid structure represents an MDP state. During navigation, the robot occasionally slips into neighboring cells upon taking a step along one of the 4 cardinal directions. We consider two tasks: `pick_n_drop` and `patrolABCD`. Figures 1a and 2a show the color-coded ground-truth labeling function. For example, the blue cell in Figure 1a denotes the pickup location and the red cells in Figure 2a represents proposition A. These labels ($\mathrm{A}, \mathrm{B}, \mathrm{C}$, etc.) could in principle denote an area that has certain properties (cold/hot), or contains landmarks (coffee/mail). It is important to emphasize that the labeling of these cells is hidden from our algorithm.



(a) (b) (c)

Fig. 1: (a) The warehouse grid world. (b) The pick and drop reward machine. (c) Solution count at increasing depths. The shaded area represents ± one standard deviation. Negative region is cut-off.

5.1 Task 1: `pick_n_drop`

The robot's objective is to perform a standard warehouse automation task: visit the pickup location (bottom right, Figure 1a) and then the drop-off location (top left, Figure 1a) in a cyclic manner while avoiding passing through a danger zone (the 4 states colored yellow in the bottom middle, Figure 1a). The corresponding ground-truth reward machine is shown in Figure 1b. Above each edge between two nodes, there is a tuple showing the label initiating the transition and the corresponding reward value. Critically, our learning algorithm does not have access to the reward machine transitions or the underlying warehouse arrangement. It operates solely on raw state trajectories extracted from the expert's patrolling policy. Using a depth-9 expert history policy, our algorithm returns 12 solutions and successfully recovers the ground-truth reward machine. It correctly assigns distinct labels to the pickup, drop-off and avoid locations while grouping all remaining states under a common label. These solutions differ only by renaming, which does not change the task specification, making them equivalent to the ground truth. To evaluate the performance of our active extension algorithm, we start with a burn-in depth of $l = 3$. Due to the sparsity of negative examples at

this depth, the number of feasible labeled reward machine models (i.e. feasible solutions to Problem 1) exceeds $150K$. We only keep $10K$ of these solutions. We run our active extension algorithm with $N_{\text{active}} = \{50, 100\}$ and a budget $\mathfrak{B} = 250$. We compare our active extension algorithm against a baseline that randomly generates feasible state trajectory pairs and queries the history policy. Results in Figure 1c show the mean solution count across 15 independent trials. With $N_{\text{active}} = 100$, our active extension algorithm converges to the ground-truth solution set in 100% of the trials by depth 12. Running our algorithm with $N_{\text{active}} = 50$ also performed well (stabilizing at the ground truth solution by depth 18), while the random baseline failed to find any restrictive constraints even as far as depth 20.[3]

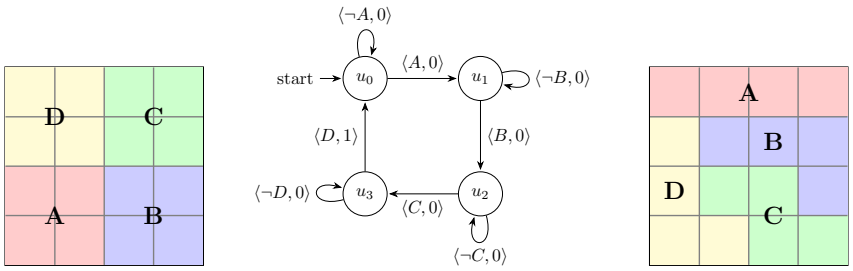
5.2 Task 2: `patrolABCD`

The robot's objective here is to patrol the rooms in the order A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D (Figure 2a). The task is encoded by the ground truth reward machine shown in Figure 2b. By using the depth-9 restriction of the expert's history policy, we recover the ground truth labeled reward machine model and the clustering of grid cells into their respective propositions up-to-renaming (this amounts to a total of **36** solutions[4]). As shown in Table 1, this results in $|\mathcal{E}_9^-| \approx 414\text{M}$, meaning that we have over 414M negative examples. In our experiments, we group these negative examples by their terminal state and sample 5000 of these negative examples at random from each group. We also test our framework on a tetris variant of the room structures (Figure 2c), to which the results remain unchanged. That is, our algorithm perfectly recovers the labeling function up to renaming.

For our active learning algorithm, we initialize the process with a burn-in depth of $l = 6$. At this depth, the history policy consists of 6895 unique branches, and solving $\text{SAT}_{l=6}$ yields a hypothesis space of 1152 distinct solutions. We constrain the negative example query budget to $\mathfrak{B} = 250$ per depth increment. We examine two sub-sampling sizes, $N_{\text{active}} \in \{100, 200\}$. Figure 3a shows the mean solution count across 30 independent trials. The blue curve represents the solution count when using the full depth-l restriction of the history policy, for $6 \leq l \leq 13$. Both $N_{\text{active}} = 100$ and $N_{\text{active}} = 200$ exhibit similar performance, achieving faster convergence to the ground truth solution set than the random sampling baseline. Specifically, with $N_{\text{active}} = 200$, 96.6% of trials converge to the ground-truth solution set (up-to-renaming) by depth 13, while 83.3% converge at the same depth with $N_{\text{active}} = 100$. In contrast, the random sampling baseline still averages 378.0 candidate solutions and exhibits a standard deviation nearly two orders of magnitude larger than that of our active extension method.

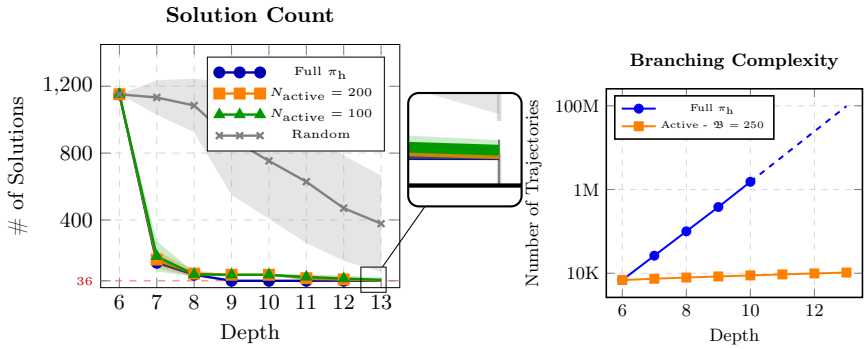
[3] We note that any depth beyond 10 is practically infeasible to solve using the full restriction of the history policy, highlighting the computational significance of our algorithm.

[4] There is 6 possible node naming permutations of the labeled reward machine model and 6 naming permutations of the labeling assignments given that the label of the first state is anchored (Equation 7c), thus we have $6 \times 6 = 36$ total renaming solutions.



(a) (b) (c)

Fig. 2: (a) The room grid world. (b) The patrol reward machine. (c) The Tetris rooms grid world.



(a) (b)

Fig. 3: (a): Reduction in solution set size vs depth. (b): Growth of the number of trajectories vs depth.

Beyond exact recovery, we show in Table 1 the efficiency gains of the active extension algorithm as compared to the full depth restriction history policy (exhaustive). The latter quickly hits a memory bottleneck, requiring approximately 24.76 GB to store over 414 M negative examples at depth 9. This is due to the exponential growth in the number of state trajectories depicted in Figure 3b. In contrast, the active method selectively queries the environment and maintains only 10.3 K branches and 0.292 M negative examples even at depth 13, reducing the memory required for negative examples to just 0.147 GB, effectively reducing the memory requirement for negative examples by two orders of magnitude.

The active extension algorithm also yields significant runtime improvements. As shown in Table 2, the exhaustive baseline is dominated by SAT solving, with a mean SAT time exceeding 7100 seconds, despite terminating at depth 9. In contrast, the active method substantially reduces the SAT burden to 2417.12 seconds on average while scaling to a deeper horizon ($l = 13$). Overall, the active

Method	**Depth** (l)	$\lvert\tau\rvert$	$\lvert\mathcal{E}_l^-\rvert$	**size** ($\lvert\tau\rvert$)	**size** ($\lvert\mathcal{E}_l^-\rvert$)
Exhaustive	9	382K	414M	0.3Gb	24.76Gb
Active Extension	13	10.3K	0.292M	0.046Gb	0.147Gb

Table 1: Memory Requirements for exhaustive vs. active search. $|\tau|$ is the total number of trajectories. $|\mathcal{E}_l^-|$ is the number of negative examples. **size** ($|\tau|$) is the memory required to store the trajectories. **size** ($|\mathcal{E}_l^-|$) is the memory required to store the negative examples.

extension achieves a mean total runtime of 3544.76 seconds, nearly a 2× speedup over the exhaustive baseline, while still recovering the full ground-truth solution set. These results demonstrate that active extension alleviates both memory and computational bottlenecks, which lays the groundwork for scalable reward machine inference from raw state trajectories.

Method	**Mean Discovery Time (s)**	**Mean SAT Time (s)**	**Total Time (s)**	**Max Depth**	**Sols**
Active Extension	1127.64 ± 186.11	2417.12 ± 6.16	3544.76 ± 188.71	13	36
Exhaustive	26.80	7158.73 ± 1640.39	7185.53 ± 1640.39	9	36

Table 2: Computation requirements for active extension vs. exhaustive Baseline. **Mean Discovery Time (s)** refers to the time spent finding the negative example set. **Mean SAT Time (s)** refers to the time spent solving a SAT problem instance. Results are over 10 separate trials. The mean SAT time for active extension includes 2350 s spent solving SAT with the depth-6 policy.

6 Discussion and conclusion

In this paper, we studied reward machine inference in an information-scarce setting where only raw state trajectories and a depth-limited history policy are available, without observing rewards, labels, or automaton nodes. We introduced a SAT-based formulation that jointly infers the reward machine transition structure and a labeling function, and established a sufficient depth condition under which additional history does not further constrain feasibility. Building on this, we proposed an active extension strategy that selectively queries informative trajectory pairs to reduce the hypothesis space efficiently, yielding substantial memory and runtime gains while still recovering the ground-truth solution set up to renaming whenever this is possible with the exhaustive version.

We emphasize that the present framework should be viewed as a foundational step, establishing the identifiability and algorithmic backbone of the problem, rather than a complete end-to-end practical solution. Because the current formulation assumes a discrete state-action space and access to the history policy induced by the optimal product policy, important questions remain regarding

robustness to finite data and scalability to richer robotic domains. Nevertheless, the framework suggests several concrete directions for improving applicability: the history policy could be estimated from state-action trajectories using statistical extensions robust to estimation error as is done in known labeling function case [12], while selective querying could substantially reduce the computational burden where exhaustive expansion is infeasible. Furthermore, extending the framework beyond the tabular setting will likely involve learning labeling functions directly from perceptual representations, potentially leveraging pre-trained models [38]. A remaining limitation, however, is that the current termination criterion requires finding solutions up to renaming, which may be unnecessarily restrictive in practice. In general, it might be possible to terminate, even earlier that the sufficient depth, if all the candidate labeled reward machine models are policy-equivalent. Integrating such an equivalence-testing procedure into the active extension process while preserving the framework's computational advantages represents a promising path for future research.

More broadly, our work fits within a larger question of why memory is needed in sequential decision-making. Arguably, there are two primary sources of such memory requirements. The first is partial observation or epistemic uncertainty, where memory is needed to construct a sufficient information state for decision-making. The second is task structure, where successful behavior depends on remembering progress through a temporally extended objective, as in our setting. Our contribution is aimed at uncovering this second form of memory structure, represented as a reward machine and labeling function, directly from policy data. A natural next step, therefore, is to relate the present framework to the literature on information states [39] and learning Partially Observed Markov Decision Processes [40], with the broader goal of developing a unified understanding of memory requirements in robot decision-making.

Acknowledgments. This work is supported in part by ONR CLEVR-AI MURI (#N00014- 21-1-2431). Antoine thanks his son Mathéo for giving him the time to finish this article. LLMs (ChatGPT, Gemini) have been used to polish parts of the writing of this paper. The output of the LLM is then thoroughly examined by the authors to maintain consistency and accuracy.

References

1. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
2. A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 1–2, pp. 41–77, 2003.
3. H. Kress-Gazit, G. Fainekos, and G. J. Pappas, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 211–236, 2018.
4. J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *Proceedings of the 34th International Conference on Machine Learning*, 2017.

5. R. Toro Icarte, T. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018.
6. R. Toro Icarte and S. A. McIlraith, "Learning reward machines for partially observable reinforcement learning," in *Advances in Neural Information Processing Systems*, 2019.
7. A. Camacho, J. Varley, A. Zeng, D. Jain, A. Iscen, and D. Kalashnikov, "Reward machines for vision-based robotic manipulation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 14 284–14 290.
8. D. DeFazio, Y. Hayamizu, and S. Zhang, "Learning quadruped locomotion policies using logical rules," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 142–150.
9. M. Baert, E. Malomgré, S. Leroux, and P. Simoens, "Reward machine inference for robotic manipulation," in *IBRL @ RLC 2025*, 2025.
10. D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," *arXiv preprint arXiv:1606.06565*, 2016.
11. Z. Xu, A. Gawel, K. Muller, M. Yan, D. Juan, and Y. Hu, "Deep reward machine learning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
12. M. L. Shehab, A. Aspeel, and N. Ozay, "Learning reward machines from partially observed policies," *Transactions on Machine Learning Research*, 2025.
13. B. Araki, K. Vodrahalli, T. Leech, C.-I. Vasile, M. D. Donahue, and D. L. Rus, "Learning to plan with logical automata," *Robotics: Science and Systems Foundation*, 2019.
14. Z. Xu, I. Gavran, Y. Ahmad, R. Majumdar, D. Neider, U. Topcu, and B. Wu, "Joint inference of reward machines and policies for reinforcement learning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 590–598.
15. R. T. Icarte, T. Q. Klassen, R. Valenzano, M. P. Castro, E. Waldie, and S. A. McIlraith, "Learning reward machines: A study in partially observable reinforcement learning," *Artificial Intelligence*, vol. 323, p. 103989, 2023.
16. J. Hu, Y. Paliwal, H. Kim, Y. Wang, and Z. Xu, "Reinforcement learning with predefined and inferred reward machines in stochastic games," *Neurocomputing*, vol. 599, p. 128170, 2024.
17. A. Abate, Y. Almulla, J. Fox, D. Hyland, and M. Wooldridge, "Learning task automata for reinforcement learning using hidden markov models," in *ECAI 2023*. IOS Press, 2023, pp. 3–10.
18. D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987.
19. Z. Xu, B. Wu, A. Ojha, D. Neider, and U. Topcu, "Active finite reward automaton inference and reinforcement learning using queries and counterexamples," in *Machine Learning and Knowledge Extraction*, A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, Eds. Cham: Springer International Publishing, 2021.
20. F. Memarian, Z. Xu, B. Wu, M. Wen, and U. Topcu, "Active task-inference-guided deep inverse reinforcement learning," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 1932–1938.
21. H. Hasanbeig, N. Y. Jeppu, A. Abate, T. Melham, and D. Kroening, "Symbolic task inference in deep reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 80, pp. 1099–1137, 2024.

22. M. Hasanbeig, N. Y. Jeppu, A. Abate, T. Melham, and D. Kroening, "DeepSynth: Automata synthesis for automatic task segmentation in deep reinforcement learning," in *AAAI*, vol. 35, no. 9, 2021, pp. 7647–7656.
23. D. Furelos-Blanco, M. Law, A. Russo, K. Broda, and A. Jonsson, "Induction of subgoal automata for reinforcement learning," in *AAAI*, vol. 34, no. 04, 2020.
24. M. Baert, S. Leroux, and P. Simoens, "Reward machine inference for robotic manipulation," *arXiv preprint arXiv:2412.10096*, 2024.
25. A. Li, Z. Chen, T. Klassen, P. Vaezipoor, R. Toro Icarte, and S. McIlraith, "Reward machines for deep rl in noisy and uncertain environments," *Advances in Neural Information Processing Systems*, vol. 37, pp. 110 341–110 368, 2024.
26. R. Parac, L. Nodari, L. Ardon, D. Furelos-Blanco, F. Cerutti, and A. Russo, "Learning robust reward machines from noisy labels," *arXiv preprint arXiv:2408.14871*, 2024.
27. C. K. Verginis, C. Koprulu, S. Chinchali, and U. Topcu, "Joint learning of reward machines and policies in environments with partially known semantics," *Artificial Intelligence*, vol. 333, p. 104146, 2024.
28. B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008.
29. M. L. Shehab, A. Aspeel, N. Arechiga, A. Best, and N. Ozay, "Learning true objectives: Linear algebraic characterizations of identifiability in inverse reinforcement learning," in *L4DC*. PMLR, 2024, pp. 1266–1277.
30. S. A. Cook, "The complexity of theorem-proving procedures," *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, 1971.
31. A. Biere, M. J. Heule, H. van Maaren, and T. Walsh, *Handbook of satisfiability*. IOS press, 2009, vol. 185.
32. H. Cao, S. Cohen, and L. Szpruch, "Identifiability in inverse reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, 2021.
33. D. Sadigh, A. D. Dragan, S. S. Sastry, and S. A. Seshia, "Active preference-based learning of reward functions," in *RSS*, Jul. 2017.
34. E. Biyik and D. Sadigh, "Batch active preference-based learning of reward functions," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 519–528. [Online]. Available: https://proceedings.mlr.press/v87/biyik18a.html
35. E. Biyik, M. Palan, N. C. Landolfi, and D. Sadigh, "Asking easy questions: A user-friendly approach to active reward learning," in *Proceedings of the 3rd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 1177–1194. [Online]. Available: https://proceedings.mlr.press/v100/biyik20a.html
36. S. Dasgupta, "Analysis of a greedy active learning strategy," in *Advances in Neural Information Processing Systems 17 (NeurIPS 2004)*, 2004, pp. 337–344.
37. R. Motwani and P. Raghavan, "Randomized algorithms," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 33–37, 1996.
38. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
39. B. Sakcak, K. G. Timperi, V. Weinstein, and S. M. LaValle, "A mathematical characterization of minimally sufficient robot brains," *The International Journal of Robotics Research*, vol. 43, no. 9, pp. 1342–1362, 2024.
40. S. Shaw, T. Manderson, C. Kessens, and N. Roy, "Toward learning pomdps beyond full-rank actions and state observability," *arXiv preprint arXiv:2601.18930*, 2026.