Hierarchies of Reward Machines

Daniel Furelos-Blanco
Imperial College London
d.furelos-blanco@imperial.ac.uk

Mark Law
ILASP Limited
mark@ilasp.com

Anders Jonsson
Universitat Pompeu Fabra
anders.jonsson@upf.edu

Krysia Broda
Imperial College London
k.broda@imperial.ac.uk

Alessandra Russo
Imperial College London
a.russo@imperial.ac.uk

Abstract

Reward machines (RMs) are a recent formalism for representing the reward function of a reinforcement learning task through a finite-state machine whose edges encode landmarks of the task using high-level events. The structure of RMs enables the decomposition of a task into simpler and independently solvable subtasks that help tackle long-horizon and/or sparse reward tasks. We propose a formalism for further abstracting the subtask structure by endowing an RM with the ability to call other RMs, thus composing a hierarchy of RMs (HRM). We exploit HRMs by treating each call to an RM as an independently solvable subtask using the options framework, and describe a curriculum-based method to induce HRMs from example traces observed by the agent. Our experiments reveal that exploiting a handcrafted HRM leads to faster convergence than with a flat HRM, and that learning an HRM is more scalable than learning an equivalent flat HRM.

1 Introduction

Around a decade ago, Dietterich et al. [15] argued the need to "learn at *multiple time scales* simultaneously, and with a rich *structure* of events and durations". Finite-state machines (FSMs) are a simple yet powerful formalism for abstractly representing temporal tasks in a structured manner. One of the most prominent types of FSMs used in reinforcement learning [RL, 46] are reward machines [RMs, 49, 51]. RMs are FSMs where each edge is labeled with (i) a propositional formula over a set of high-level events that capture a task's landmark/subgoal, and (ii) a reward for satisfying the formula. Hence, RMs fulfill the need for structuring events and durations. Hierarchical reinforcement learning [HRL, 6] frameworks, such as the options framework [48], have been applied over RMs to learn policies at two levels of abstraction: (i) which formula to take at a given RM state in order to accomplish the task, and (ii) which action to take in order to accomplish the chosen formula [17, 49]. Therefore, these structures also allow learning at multiple scales simultaneously. The decomposition into simple subtasks powered by HRL algorithms enables handling long-horizon and sparse reward tasks more easily. Furthermore, RMs can be used to address partial observability by acting as an external memory that keeps track of the subgoals achieved so far and those to be achieved.

In this work, we enhance the abstraction power of RMs by defining *hierarchies of RMs (HRMs)*, where constituent RMs can invoke other RMs (Section 3). We prove that an arbitrary HRM can be transformed into an *equivalent* flat HRM that behaves exactly like the original RMs, and show that, under certain conditions, the equivalent flat representation can have exponentially more states and edges. We propose to apply HRL algorithms to HRMs by treating each call to an RM as a subtask, akin to previous works on RMs (Section 4). Learning policies in HRMs enhances the properties posed by Dietterich et al. since (i) there is an arbitrary number of time scales to learn across (and not only two), and (ii) there is a richer and more diverse range of increasingly abstract events and durations. An

Preprint. Under review.

additional key benefit of using hierarchies is *modularity*; that is, a given RM and its associated policies are *reusable* across several HRMs. Finally, we introduce LHRM, a curriculum-based method for *learning* HRMs from traces given a set of hierarchically composable tasks (Section 5). Empirically, we show that leveraging a handcrafted HRM enables faster convergence than an equivalent flat HRM in complex tasks. Our experiments also reveal that learning an HRM (or, more specifically, its root) is more efficient than learning an equivalent flat HRM since (i) the constituent RMs are smaller (i.e., they have fewer states and edges), and (ii) previously learned RMs can be used to efficiently *explore* the environment in the search for traces in new, more complex tasks.

2 Background

Given a finite set X, we use $\Delta(X)$ to denote the probability simplex over X, X^* to denote (possibly empty) sequences of elements from X, and X^+ to denote non-empty sequences. We also use $\perp$ and $\top$ to denote the truth values false and true, respectively. $\mathbb{1}[A]$ is the indicator function of event A.

We represent reinforcement learning [RL, 46] tasks as episodic labeled Markov decision processes (MDPs) [55], defined as tuples $\mathcal{M} = \langle \Sigma, A, p, r, \gamma, \mathcal{P}, \mathcal{L}, \tau \rangle$, where Σ is a set of states, A is a set of actions, $p : \Sigma \times A \to \Delta(\Sigma)$ is a transition function, $r : (\Sigma \times A)^+ \times \Sigma \to \mathbb{R}$ is a reward function, $\gamma \in [0, 1)$ is a discount factor, $\mathcal{P}$ is a finite set of *propositions*, $\mathcal{L} : \Sigma \to 2^{\mathcal{P}}$ is a *labeling function* mapping states to proposition subsets (or *labels*), and $\tau : (\Sigma \times A)^* \times \Sigma \to \{\perp, \top\}^2$ is a termination function. Hence the transition function p is Markovian, but the reward function r and termination function τ are not. Given a *history* $h_t = \langle \sigma_0, a_0, \ldots, \sigma_t \rangle \in (\Sigma \times A)^* \times \Sigma$, a *label trace* (or trace, for short) $\lambda_t = \langle \mathcal{L}(\sigma_0), \ldots, \mathcal{L}(\sigma_t) \rangle \in (2^P)^+$ assigns labels to all states in h_t. We assume that (λ_t, σ_t) captures all relevant information about h_t; thus, the reward and transition information can be written $r(h_t, a_t, \sigma_{t+1}) = r(h_{t+1}) = r(\lambda_{t+1}, \sigma_{t+1})$ and $\tau(h_t) = \tau(\lambda_t, \sigma_t)$, respectively. We aim to find a policy $\pi : (2^{\mathcal{P}})^+ \times \Sigma \to A$, a mapping from traces-states to actions, which maximizes the expected cumulative discounted reward (or *return*) $R_t = \mathbb{E}_\pi[\sum_{k=t}^{n} \gamma^{k-t} r_t]$, where n is the last episode's step.

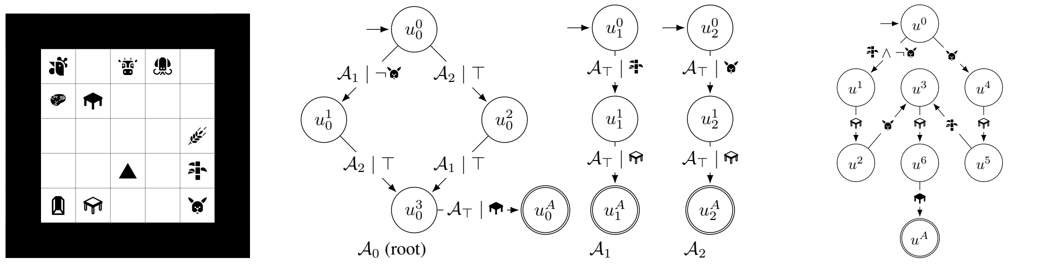
At time t, the trace is $\lambda_t \in (2^{\mathcal{P}})^+$, and the agent observes a tuple $\boldsymbol{\sigma}_t = \langle \sigma_t^\Sigma, \sigma_t^T, \sigma_t^G \rangle$, where $\sigma_t^\Sigma \in \Sigma$ is the state and $(\sigma_t^T, \sigma_t^G) = \tau(\lambda_t, \sigma_t^\Sigma)$ is the termination information, with σ_t^T and σ_t^G indicating whether or not the history $(\lambda_t, \sigma_t^\Sigma)$ is terminal or a goal, respectively. If the history is non-terminal, the agent runs action $a_t \in A$, and the environment transitions to state $\sigma_{t+1}^\Sigma \sim p(\cdot | \sigma_t^\Sigma, a_t)$. The agent then extends the trace as $\lambda_{t+1} = \lambda_t \oplus \mathcal{L}(\sigma_{t+1}^\Sigma)$, receives reward $r_{t+1} = r(\lambda_{t+1}, \sigma_{t+1}^\Sigma)$, and observes a new tuple $\boldsymbol{\sigma}_{t+1}$. A trace λ_t is a *goal* trace if $\sigma_t^T = \top \wedge \sigma_t^G = \top$, a *dead-end* trace if $\sigma_t^T = \top \wedge \sigma_t^G = \perp$, and an *incomplete* trace if $\sigma_t^T = \perp$. We assume that the reward is $r(\lambda_{t+1}, \sigma_{t+1}^\Sigma) = \mathbb{1}[\tau(\lambda_{t+1}, \sigma_{t+1}^\Sigma) = (\top, \top)]$, i.e. 1 for goal histories and 0 otherwise.

A *reward machine* [RM, 49, 51] is a tuple $\langle U, \mathcal{P}, \varphi, r, u^0, U^A, U^R \rangle$, where U is a finite set of states; $\mathcal{P}$ is a finite set of propositions; $\varphi : U \times U \to \text{DNF}_{\mathcal{P}}$ is a state transition function such that $\varphi(u, u')$ denotes the disjunctive normal form (DNF) formula over $\mathcal{P}$ that must be satisfied to transition from u to u'; $r : U \times U \to \mathbb{R}$ is a reward transition function such that $r(u, u')$ is the reward for transitioning from u to u'; $u^0 \in U$ is an initial state; $U^A \subseteq U$ is a set of accepting states; and $U^R \subseteq U$ is a set of rejecting states. Ideally, RM states should capture traces, such that pairs (u, σ) of an RM state and an MDP state are sufficient to predict the future, and such that the reward $r(u, u')$ corresponds to the reward obtained in the underlying MDP. The state transition function is *deterministic*, i.e. at most one formula from each state is satisfied. To verify whether a formula is satisfied by a label $L \subseteq \mathcal{P}$, L is used as truth assignment where propositions $p \in L$ are true, and false otherwise (e.g., $\{a\} \models a \wedge \neg b$).

The *options* framework [48] addresses temporal abstraction in RL. Given an episodic labeled MDP, an option is a tuple $\omega = \langle I_\omega, \pi_\omega, \beta_\omega \rangle$, where $I_\omega \subseteq \Sigma$ is the option's initiation set, $\pi_\omega : \Sigma \to A$ is the option's policy, and $\beta_\omega : \Sigma \to [0, 1]$ is the option's termination condition. An option is available in $\sigma \in \Sigma$ if $\sigma \in I_\omega$, selects actions according to π_ω, and terminates in $\sigma \in \Sigma$ with probability $\beta_\omega(\sigma)$.

3 Formalization of Hierarchies of Reward Machines

In what follows we introduce our formalism for hierarchically composing reward machines, and propose the CraftWorld domain (cf. Figure 1a) to illustrate it. In this domain, the agent (▲) can move forward or rotate 90°, staying put if it moves towards a wall. Grid locations are labeled with propositions from $\mathcal{P} = \{$📕, 🍄, 🐮, 🪧, 🌾, 🐔, 🥩, 🐰, 🦑, 🛠$\}$. The agent observes propositions that it steps


u_0^0
$\mathcal{A}_1 \mid \neg$🐰
$\mathcal{A}_2 \mid \top$
u_0^1
u_0^2
$\mathcal{A}_2 \mid \top$
$\mathcal{A}_1 \mid \top$
u_0^3
$\mathcal{A}_\top \mid$ ■
u_0^A
$\mathcal{A}_0$ (root)
u_1^0
$\mathcal{A}_\top \mid$ 🎋
u_1^1
$\mathcal{A}_\top \mid$ □
u_1^A
$\mathcal{A}_1$
u_2^0
$\mathcal{A}_\top \mid$ 🐰
u_2^1
$\mathcal{A}_\top \mid$ □
u_2^A
$\mathcal{A}_2$
u^0
🎋 $\wedge\, \neg$🐰
🐰
u^1
u^3
u^4
□
🐰
□
🎋
□
u^2
u^6
u^5
■
u^A

(a) (b) (c)

Figure 1: A CRAFTWORLD grid (a), an HRM for BOOK (b), and an equivalent flat HRM (c). An edge from state u to u' of an RM $\mathcal{A}_i$ is of the form $\mathcal{A}_j \mid \varphi_i(u, u', \mathcal{A}_j)$, double circled states are accepting states, and loop transitions are omitted. Calls to the leaf machine $\mathcal{A}_\top$ are omitted in (c).

Table 1: List of CRAFTWORLD tasks. Descriptions "x ; y" express sequential order (observe/do x then y), and descriptions "x & y" express that x and y can be observed/done in any order.

Task	h	Description	Task	h	Description	Task	h	Description
BATTER	1	(🌾 & 🐓) ; ■	QUILL	1	(🦑 & 🐓) ; ■	BOOKQUILL	3	BOOK & QUILL
BUCKET	1	🪣 ; ■	SUGAR	1	🎋 ; ■	MILKB.SUGAR	3	MILKBUCKET & SUGAR
COMPASS	1	(🪣 & 🥩) ; □	BOOK	2	(PAPER & LEATHER) ; ■	CAKE	4	BATTER ; MILKB.SUGAR ; □
LEATHER	1	🐰 ; □	MAP	2	(PAPER & COMPASS) ; ■			
PAPER	1	🎋 ; □	MILKBUCKET	2	BUCKET ; 🐮			

on (e.g., {🐓} in the top-left corner). Table 1 lists several tasks that consist of observing a sequence of propositions, where the reward is 1 if the sequence is observed and 0 otherwise. These tasks are based on those from similar domains [2, 49], but they can be defined in terms of each other.

Reward machines (RMs) are the building blocks of our formalism. To constitute a hierarchy of RMs, we need to endue RMs with the ability to call each other. We redefine the *state transition function* as $\varphi : U \times U \times \mathcal{A} \to \text{DNF}_\mathcal{P}$, where $\mathcal{A}$ is a set of RMs. The expression $\varphi(u, u', \mathcal{A}_j)$ denotes the disjunctive normal form (DNF) formula over $\mathcal{P}$ that must be satisfied to transition from $u \in U$ to $u' \in U$ by calling RM $\mathcal{A}_j \in \mathcal{A}$. We refer to the formulas $\varphi(u, u', \mathcal{A}_j)$ as *contexts* since they represent conditions under which calls are made. As we will see later, contexts help preserve determinism and must be satisfied to start a call (a necessary but not sufficient condition). The hierarchies we consider contain an RM $\mathcal{A}_\top$ called the *leaf* RM, which solely consists of an accepting state (i.e., $U_\top = U_\top^A = \{u_\top^0\}$), and immediately returns control to the RM that calls it.

Definition 1. A *hierarchy of reward machines (HRM)* is a tuple $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$, where $\mathcal{A} = \{\mathcal{A}_0, \ldots, \mathcal{A}_{m-1}\} \cup \{\mathcal{A}_\top\}$ is a set of m RMs and the leaf RM $\mathcal{A}_\top$, $\mathcal{A}_r \in \mathcal{A} \setminus \{\mathcal{A}_\top\}$ is the root RM, and $\mathcal{P}$ is a finite set of propositions used by all constituent RMs.

We make the following *assumptions*: (i) HRMs do not have circular dependencies (i.e., an RM cannot be called back from itself, including recursion), (ii) rejecting states are global rejectors (i.e., cause the root task to fail), (iii) accepting and rejecting states do not have transitions to other states, and (iv) the reward function of the root corresponds to the reward obtained in the underlying MDP. Given assumption (i), each RM $\mathcal{A}_i$ has a *height* h_i, which corresponds to the maximum number of nested calls needed to reach the leaf. Formally, if $i = \top$, then $h_i = 0$; otherwise, $h_i = 1 + \max_j h_j$, where j ranges over all RMs called by $\mathcal{A}_i$ (i.e., there exists $(u, v) \in U_i \times U_i$ such that $\varphi_i(u, v, \mathcal{A}_j) \neq \perp$).

Figure 1b shows BOOK's HRM, which consists of a root and RMs for PAPER and LEATHER. The root has height 2 and shows that the subtasks can be run in any order followed by observing ■, whereas the called RMs have height 1 and consist of observing a two proposition sequence.

In the following paragraphs we describe how an HRM processes a label trace. To indicate where the agent is in an HRM, we define the notion of hierarchy states.

Definition 2. Given an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$, a *hierarchy state* is a tuple $\langle \mathcal{A}_i, u, \Psi, \Gamma \rangle$, where $\mathcal{A}_i \in \mathcal{A}$ is an RM, $u \in U_i$ is a state, $\Psi \in \text{DNF}_\mathcal{P}$ is an accumulated context, and Γ is a call stack.

Definition 3. Given an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$, a *call stack* Γ contains tuples $\langle u, v, \mathcal{A}_i, \mathcal{A}_j, \phi, \Psi \rangle$, each corresponding to a call where $u \in U_i$ is the state from which the call is made; $v \in U_i$ is the next state in the calling RM $\mathcal{A}_i$ once an accepting state of the called RM $\mathcal{A}_j \in \mathcal{A}$ is reached; $\phi \in \text{DNF}_\mathcal{P}$ are the disjuncts of $\varphi_i(u, v, \mathcal{A}_j)$ satisfied by a label; and $\Psi \in \text{DNF}_\mathcal{P}$ is the accumulated context.

Call stacks determine where to resume the execution. Each RM appears in the stack at most once since, by assumption, HRMs have no circular dependencies. We use $\Gamma \oplus \langle u, v, \mathcal{A}_i, \mathcal{A}_j, \phi, \Psi\rangle$ to denote a stack recursively defined by a stack Γ and a top element $\langle u, v, \mathcal{A}_i, \mathcal{A}_j, \phi, \Psi\rangle$, where the *accumulated context* Ψ is the condition under which a call from a state u is made. The initial hierarchy state of an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$ is $\langle \mathcal{A}_r, u_r^0, \top, []\rangle$: we are in the initial state of the root, there is no accumulated context, and the stack is empty.

At the beginning of this section, we mention that satisfying the context of a call is a necessary but not sufficient condition to start the call. We now introduce a sufficient condition, called exit condition.

Definition 4. Given an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$ and a hierarchy state $\langle \mathcal{A}_i, u, \Psi, \Gamma\rangle$, the *exit condition* $\varPhi_{i,u,\Psi} \in \text{DNF}_\mathcal{P}$ is the formula that must be satisfied in order to leave that hierarchy state. Formally,

$$\varPhi_{i,u,\Psi} = \begin{cases} \Psi & \text{if } i = \top, \\ \bigvee_{\substack{\phi=\varphi_i(u,v,\mathcal{A}_j), \\ \phi\neq\bot, v\in U_i, \mathcal{A}_j\in\mathcal{A}}} \varPhi_{j,u_j^0,\text{DNF}(\Psi\wedge\phi)} & \text{otherwise,} \end{cases}$$

where $\text{DNF}(\Psi \wedge \phi)$ is $\Psi \wedge \phi$ in DNF. The formula is Ψ if $\mathcal{A}_i$ is $\mathcal{A}_\top$ since it always returns control once called. Otherwise, the formula is recursively defined as the disjunction of the exit conditions from the initial state of the called RM. For instance, the exit condition for the initial hierarchy state in Figure 1b is $\{$🎋$\wedge \neg$🐮$,$ 🐮$\}$.

We now have everything needed to define the *hierarchical transition function* $\delta_\mathcal{H}$, which maps a hierarchy state $\langle \mathcal{A}_i, u, \Psi, \Gamma\rangle$ into another given a label L. There are three cases:

1. If u is an accepting state of $\mathcal{A}_i$ and the stack Γ is non-empty, pop the top element of Γ and return control to the previous RM, recursively applying $\delta_\mathcal{H}$ in case several accepting states are reached simultaneously. Formally, the next hierarchy state is $\delta_\mathcal{H}(\langle \mathcal{A}_j, u', \top, \Gamma'\rangle, \bot)$ if $u \in U_i^A$, $|\Gamma| > 0$, where $\Gamma = \Gamma' \oplus \langle \cdot, u', \mathcal{A}_j, \mathcal{A}_i, \cdot, \cdot\rangle$, $\bot$ denotes a label that cannot satisfy any formula, and $\cdot$ denotes something unimportant for the case.
2. If L satisfies the context of a call and the exit condition from the initial state of the corresponding called RM, push the call onto the stack and recursively apply $\delta_\mathcal{H}$ until $\mathcal{A}_\top$ is reached. Formally, the next hierarchy state is $\delta_\mathcal{H}(\langle \mathcal{A}_j, u_j^0, \Psi', \Gamma \oplus \langle u, u', \mathcal{A}_i, \mathcal{A}_j, \phi, \Psi\rangle\rangle, L)$ if $L \models \varPhi_{j,u_j^0,\Psi'}$, where $\phi = \varphi_i(u, u', \mathcal{A}_j)(L)$ and $\Psi' = \text{DNF}(\Psi \wedge \phi)$. Here, $\varphi(L)$ denotes the disjuncts of a DNF formula $\varphi \in \text{DNF}_\mathcal{P}$ satisfied by label L.
3. If none of the previous conditions holds, the hierarchy state remains unchanged.

The state transition functions φ of the RMs must be such that $\delta_\mathcal{H}$ is *deterministic*, i.e. a label cannot simultaneously satisfy the contexts and exit conditions associated with two triplets $\langle u, v, \mathcal{A}_i\rangle$ and $\langle u, v', \mathcal{A}_j\rangle$ such that either (i) $v = v'$ and $i \neq j$, or (ii) $v \neq v'$. Contexts help enforcing determinism by making formulas mutually exclusive. For instance, if the call to $\mathcal{A}_1$ from the initial state of $\mathcal{A}_0$ in Figure 1b had context $\top$ instead of $\neg$🐮, then $\mathcal{A}_1$ and $\mathcal{A}_2$ could be both started if $\{$🎋$,$ 🐮$\}$ was observed, thus making the HRM non-deterministic. Finally, we introduce hierarchy traversals, which determine how a label trace is processed by an HRM using $\delta_\mathcal{H}$.

Definition 5. Given a label trace $\lambda = \langle L_0, \ldots, L_n\rangle$, a *hierarchy traversal* $\mathcal{H}(\lambda) = \langle v_0, v_1, \ldots, v_{n+1}\rangle$ is a unique sequence of hierarchy states such that (i) $v_0 = \langle \mathcal{A}_r, u_r^0, \top, []\rangle$, and (ii) $\delta_\mathcal{H}(v_i, L_i) = v_{i+1}$ for $i = 0, \ldots, n$. An HRM $\mathcal{H}$ *accepts* λ if $v_{n+1} = \langle \mathcal{A}_r, u, \top, []\rangle$ and $u \in U_r^A$ (i.e., an accepting state of the root is reached). Analogously, $\mathcal{H}$ *rejects* λ if $v_{n+1} = \langle \mathcal{A}_k, u, \cdot, \cdot\rangle$ and $u \in U_k^R$ for any $k \in [0, m-1]$ (i.e., a rejecting state in the HRM is reached).

Example 1. The HRM in Figure 1b accepts the label trace $\lambda = \langle\{$🎋$\}, \{$⛩$\}, \{\}, \{$🐮$\}, \{$⛩$\}, \{$☗$\}\rangle$, whose traversal is $\mathcal{H}(\lambda) = \langle\langle \mathcal{A}_0, u_0^0, \top, []\rangle, \langle \mathcal{A}_1, u_1^1, \top, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg$🐮$, \top\rangle]\rangle, \langle \mathcal{A}_0, u_0^1, \top, []\rangle, \langle \mathcal{A}_0, u_0^1, \top, []\rangle, \langle \mathcal{A}_2, u_2^1, \top, [\langle u_0^1, u_0^3, \mathcal{A}_0, \mathcal{A}_2, \top, \top\rangle]\rangle, \langle \mathcal{A}_0, u_0^3, \top, []\rangle, \langle \mathcal{A}_0, u_0^A, \top, []\rangle\rangle$. Appendix B expands this example showing the application of the hierarchical transition function step by step.

The theorem below states that the behavior of an HRM can be reproduced by an *equivalent* flat HRM (i.e., an HRM whose root has height 1), which captures the original RM definition by Toro Icarte et al. [49]. Figure 1c shows a flat HRM for the BOOK task. We prove the theorem by construction in Appendix C.1 through the progressive embedding of low-height RMs into RMs with larger height.

Theorem 1. *Given an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$, there exists an equivalent flat HRM $\bar{\mathcal{H}} = \langle \bar{\mathcal{A}}, \bar{\mathcal{A}}_r, \mathcal{P} \rangle$.*

Given the construction used to prove Theorem 1, we estimate the number of states and edges that the resulting flat HRM has. These quantities, as stated by the theorem below, may grow *exponentially* in the height of the root. We prove this result in Appendix C.2 through a sample instance of a general HRM parametrization. As later shown in Section 6, the more states and edges, the harder it is to learn an HRM, thus an exponential reduction in these quantities can greatly improve learning performance.

Theorem 2. *Let $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$ be an HRM and let h_r be the height of its root $\mathcal{A}_r$. The number of states and edges in an equivalent flat HRM $\bar{\mathcal{H}}$ may be exponential in h_r.*

4 Policy Learning in Hierarchies of Reward Machines

The high-level temporal structure of an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$ is exploited using *options* [48]. Given a constituent RM $\mathcal{A}_i \in \mathcal{A}$, a state $u \in U_i$ and a context Ψ, an option $\omega_{i,u,\Psi}^{j,\phi}$ is derived for each non-false disjunct ϕ of each transition $\varphi_i(u, v, \mathcal{A}_j)$, where $v \in U_i$ and $\mathcal{A}_j \in \mathcal{A}$. An option is either (i) a *formula option* if $j = \top$ (i.e., $\mathcal{A}_\top$ is called), or (ii) a *call option* otherwise. A formula option attempts to reach a label that satisfies $\phi \wedge \Psi$ through primitive actions, whereas a call option aims to reach an accepting state of the called RM $\mathcal{A}_j$ under context $\phi \wedge \Psi$ by invoking other options. We here describe the main aspects of policy learning and discuss implementation details in Appendix D.

Policies. Policies are ϵ-greedy during training, and greedy during evaluation. A *formula option's policy* is derived from a Q-function $Q_{\phi\wedge\Psi}(\sigma^\Sigma, a; \theta_{\phi\wedge\Psi})$ approximated by a deep Q-network [DQN, 38] with parameters $\theta_{\phi\wedge\Psi}$, which outputs the Q-value of each action given an MDP state. We store all options' experiences $(\boldsymbol{\sigma}_t, a, \boldsymbol{\sigma}_{t+1})$ in a single replay buffer $\mathcal{D}$, thus performing intra-option learning [47]. The Q-learning update uses the following loss function:

$$\mathbb{E}_{(\boldsymbol{\sigma}_t, a, \boldsymbol{\sigma}_{t+1}) \sim \mathcal{D}} \left[\left(r_{\phi\wedge\Psi}(\boldsymbol{\sigma}_{t+1}) + \gamma \max_{a'} Q_{\phi\wedge\Psi}(\sigma_{t+1}^\Sigma, a'; \theta_{\phi\wedge\Psi}^-) - Q_{\phi\wedge\Psi}(\sigma_t^\Sigma, a; \theta_{\phi\wedge\Psi}) \right)^2 \right], \quad (1)$$

where $r_{\phi\wedge\Psi}(\boldsymbol{\sigma}_{t+1}) = \mathbb{1}[\mathcal{L}(\sigma_{t+1}^\Sigma) \models \phi \wedge \Psi]$, i.e. the reward is 1 if $\phi \wedge \Psi$ is satisfied and 0 otherwise; the term $Q_{\phi\wedge\Psi}(\sigma_{t+1}^\Sigma, a'; \theta_{\phi\wedge\Psi}^-)$ is 0 when $\phi \wedge \Psi$ is satisfied or a dead-end is reached (i.e., $\sigma_{t+1}^T = \top \wedge \sigma_{t+1}^G = \perp$); and $\theta_{\phi\wedge\Psi}^-$ are the parameters of a fixed target network.

A *call option's policy* is induced by a Q-function $Q_i(\sigma^\Sigma, u, \Psi, \langle \mathcal{A}_j, \phi \rangle; \theta_i)$ associated with the called RM $\mathcal{A}_i$ and approximated by a DQN with parameters θ_i that outputs the Q-value of each call in the RM given an MDP state, an RM state and a context. We store experiences $(\boldsymbol{\sigma}_t, \omega_{i,u,\Psi}^{j,\phi}, \boldsymbol{\sigma}_{t+k})$ in a replay buffer $\mathcal{D}_i$ associated with $\mathcal{A}_i$, and perform SMDP Q-learning using the following loss:

$$\mathbb{E}_{(\boldsymbol{\sigma}_t, \omega_{i,u,\Psi}^{j,\phi}, \boldsymbol{\sigma}_{t+k}) \sim \mathcal{D}_i} \left[\left(r + \gamma^k \max_{j', \phi'} Q_i(\sigma_{t+k}^\Sigma, u', \Psi', \langle \mathcal{A}_{j'}, \phi' \rangle; \theta_i^-) - Q_i(\sigma_t^\Sigma, u, \Psi, \langle \mathcal{A}_j, \phi \rangle; \theta_i) \right)^2 \right],$$

where k is the number of steps between $\boldsymbol{\sigma}_t$ and $\boldsymbol{\sigma}_{t+k}$; r is the sum of discounted rewards during this time; u' and Ψ' are the RM state and context after running the option; $\mathcal{A}_{j'}$ and ϕ' correspond to an outgoing transition from u', i.e. $\phi' \in \varphi_i(u', \cdot, \mathcal{A}_{j'})$; and θ_i^- are the parameters of a fixed target network. The term $Q_i(\sigma_{t+k}^\Sigma, \dots)$ is 0 if u' is accepting or rejecting. Following the definition of $\delta_{\mathcal{H}}$, Ψ' is $\top$ if the hierarchy state changes; thus, $\Psi' = \top$ if $u' \neq u$, and $\Psi' = \Psi$ otherwise. Following our previous assumption on the MDP reward, we define reward transition functions as $r_i(u, u') = \mathbb{1}[u \notin U_i^A \wedge u' \in U_i^A]$. Learning a call option's policy and lower-level option policies simultaneously can be unstable due to *non-stationarity* [34], e.g. the same lower-level option may only sometimes achieve its goal. To relax this problem, experiences are added to the buffer only when options achieve their local goal (i.e., call options assume low-level options to terminate successfully).

Termination. An option terminates in two cases. First, if the episode ends in a goal state or in a dead-end state. Second, if the hierarchy state changes and either successfully completes the option or interrupts the option. Concretely, a formula option $\omega_{i,u,\Psi}^{j,\phi}$ is only applicable in a hierarchy state $\langle \mathcal{A}_i, u, \Psi, \Gamma \rangle$, while a call option $\omega_{i,u,\Psi}^{j,\phi}$ always corresponds to a stack item $\langle u, \cdot, \mathcal{A}_i, \mathcal{A}_j, \phi, \Psi \rangle$. We can thus analyze the hierarchy state to see if an option is still executing or should terminate.

Algorithm. An *option stack* $\Omega_{\mathcal{H}}$ stores the currently executing options. Initially, $\Omega_{\mathcal{H}}$ is empty. At each step, $\Omega_{\mathcal{H}}$ is filled by repeatedly choosing options starting from the current hierarchy state using call option policies until a formula option is selected. Since HRMs have, by assumption, no circular dependencies, a formula option will eventually be chosen. After an action is selected using the formula option's policy and applied, the DQNs associated with formula options are updated. The new hierarchy state is then used to determine which options in $\Omega_{\mathcal{H}}$ have terminated. Experiences for the terminated options that achieved their goal are pushed into the corresponding buffers, and the DQNs associated with the call options are updated. Finally, $\Omega_{\mathcal{H}}$ is updated to match the call stack of the new hierarchy state (if needed) by mapping each call stack item into an option, and adding it to $\Omega_{\mathcal{H}}$ if it is not already there. By aligning the option stack with the call stack, we can update DQNs for options that ended up being run in *hindsight* and which would have been otherwise ignored.

5 Learning Hierarchies of Reward Machines from Traces

In the previous section, we explained how a *given* HRM can be exploited using options; however, manually engineering an HRM is not practical. We here describe LHRM, a method that interleaves policy learning with the learning of HRMs from interaction. We consider a *multi-task* setting. Given T tasks and I instances (e.g., grids) of an environment, the agent must learn through interaction (i) an HRM for each task, and (ii) general policies to reach the goal in each task-instance pair. Namely, the agent interacts with $T \times I$ labeled MDPs $\mathcal{M}_{ij}$, where $i \in [1, T]$ and $j \in [1, I]$. The learning of an HRM is performed using examples from several instances to represent a task's structure more accurately. All MDPs share the same propositions $\mathcal{P}$ and actions A, while those defined on a particular instance share the same states Σ and labeling function $\mathcal{L}$. To stabilize policy learning, we assume dead-end traces to be common across tasks.[1] For brevity, the height h of the root of a task's HRM is referred to as *task level*.

Curriculum Learning [8]. LHRM learns the HRMs for each of the tasks following a similar method to that by Pierrot et al. [42]. We assume that the level h of each task is known (see Table 1 for CraftWorld), and learning progresses from lower to higher levels. Before starting an episode, LHRM selects an MDP $\mathcal{M}_{ij}$, where $i \in [1, T]$ and $j \in [1, I]$. For each MDP $\mathcal{M}_{ij}$, an estimate of the average undiscounted return R_{ij} is kept, which determines the probability of choosing tasks and instances. A task is first selected based on the lowest return across its instances, and an instance is then chosen based on its return with respect to other instances' returns. In both cases, low returns are mapped into higher probabilities (see details in Appendix E). Initially, only level 1 MDPs can be chosen. When the minimum average return across MDPs up to the current level surpasses a threshold Δ, the current level increases by 1. Hence, LHRM ensures that the learned HRMs and their associated policies are reusable by those of higher level tasks.

Learning an HRM. The learning of an HRM is analogous to learning a flat RM [17, 21, 50, 55]. The objective is to learn the state transition function φ_r of the root $\mathcal{A}_r$ with height h_r given (i) a set of states U_r, (ii) a set of label traces $\Lambda = \Lambda^G \cup \Lambda^D \cup \Lambda^I$, (iii) a set of propositions $\mathcal{P}$, (iv) a set of RMs $\mathcal{A}$ with lower heights than h_r, (v) a set of callable RMs $\mathcal{A}_{\mathcal{C}} \subseteq \mathcal{A}$ (by default, $\mathcal{A}_{\mathcal{C}} = \mathcal{A}$), and (vi) the maximum number of disjuncts κ in the DNF formulas labeling the edges. Without loss of generality, we assume that each RM has a single accepting state and a single rejecting state; hence, U_r contains at least 3 states (the initial, accepting and rejecting states). The learned state transition function φ_r is such that the resulting HRM $\mathcal{H} = \langle \mathcal{A} \cup \{\mathcal{A}_r\}, \mathcal{A}_r, \mathcal{P} \rangle$ accepts all goal traces Λ^G, rejects all dead-end traces Λ^D, and neither accepts or rejects incomplete traces Λ^I. The transition functions can be represented as sets of logic rules, which are learned using the ILASP [30] inductive logic programming system (see Appendix F for details on the ILASP encoding).

Interleaving Algorithm. LHRM *interleaves* the induction of HRMs with policy learning akin to Furelos-Blanco et al. [17]. Initially, the HRM's root of each task $i \in [1, T]$ consists of 3 states (the initial, accepting and rejecting states) and does not accept nor reject anything. A new HRM is learned when an episode's label trace is not correctly recognized by the current HRM (i.e., if a goal trace is not accepted, a dead-end trace is not rejected, or an incomplete trace is accepted or rejected). The number of states in U_r increases by 1 when an HRM that covers the examples cannot be learned, hence guaranteeing that the root has the smallest possible number of states (i.e., it is *minimal*) for a specific value of κ. When an HRM for task i is learned, the returns R_{ij} in the curriculum are set to 0 for all $j \in [1, I]$. Analogously to some methods on learning RMs [21, 50, 55], the first HRM for a

[1]The term $Q_{\phi \wedge \Psi}(\sigma^{\Sigma}_{t+1}, \ldots)$ in Equation 1 is 0 if $\sigma^{T}_{t+1} = \top \wedge \sigma^{G}_{t+1} = \bot$. Since experiences $(\boldsymbol{\sigma}_t, a, \boldsymbol{\sigma}_{t+1})$ are shared through the replay buffer, evaluating the condition differently can produce instabilities.

task is learned using a set of traces; specifically, the ρ_s shortest traces from a set of ρ goal traces are used. Empirically, short traces speed up HRM learning. Finally, LHRM leverages learned options to *explore* the environment during the collection of the ρ goal traces, speeding up the process when labels are sparse. An option is selected from a pool formed by options appearing in RMs with lower heights, its greedy policy is run until termination.

6 Experimental Results

We evaluate the components of our approach using two domains. We report the average performances across 5 runs, each consisting of a different set of 10 random instances. Learning curves show the average undiscounted return obtained by the greedy policy every 100 episodes across instances. We present averages and standard errors (in brackets) for other metrics (e.g., learning times). In HRM learning experiments, we set a 2-hour timeout to learn the HRMs. See Appendix G for experimental details and extended results.

Domains. We consider four grid types for the CRAFTWORLD domain introduced in Section 3: an open plan 7×7 grid (OP, Figure 1a), an open plan 7×7 grid with a lava location (OPL), a 13×13 four rooms grid [FR, 48], and a 13×13 four rooms grid with a lava location per room (FRL). The lava proposition in OPL and FRL must always be avoided. WATERWORLD [27, 43, 49] consists of a 2D box containing 12 balls of 6 different colors (2 per color) moving at a constant speed in a fixed direction. The agent ball can change its velocity in any cardinal direction. The propositions $\mathcal{P} = \{r, g, b, c, m, y\}$ are the balls' colors. Labels consist of the color of the balls the agent overlaps with and, unlike CRAFTWORLD, they may contain multiple propositions. The tasks consist in observing specific color sequences. We consider two settings: without dead-ends (WOD) and with dead-ends (WD). In WD, the agent must avoid 2 balls of an additional color.

Policy Learning. We compare the performance of policy learning in *handcrafted* non-flat HRMs against minimal flat equivalents. Figure 2 shows the learning curves for some CRAFTWORLD tasks in the FRL setting. The convergence rate is similar in the simplest task (MILKBUCKET), but higher for non-flat HRMs in the hardest tasks. As both approaches use the same set of formula option policies, differences arise from the lack of modularity in flat HRMs. Call options, which are not present in flat HRMs, constitute independent modules that help reduce reward sparsity. MILKBUCKET consists of less high-level steps than BOOKQUILL and CAKE, thus reward is less sparse and non-flat HRMs are not beneficial. The effectiveness of non-flat HRMs is also limited when (i) the task's goal is always reachable regardless of the chosen options (e.g., if the are no edges to rejecting states, like in OP and FR), and (ii) instances do not make reward sparse, like in OPL (the grid is small) or WATERWORLD (the balls move and can get near the agent easily).

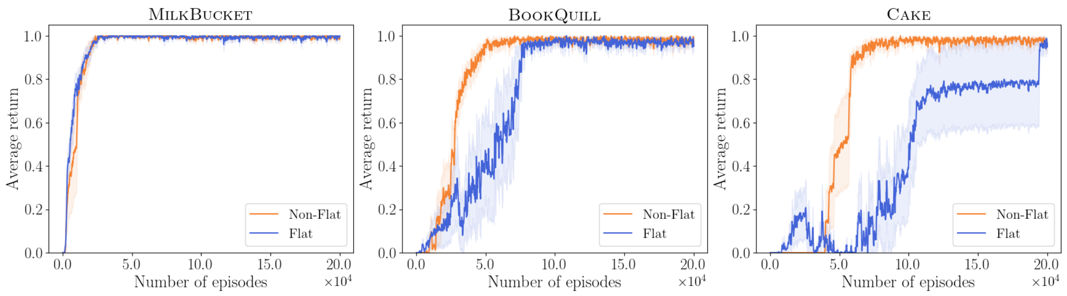


Figure 2: Learning curves for three CRAFTWORLD tasks in the FRL setting using handcrafted HRMs.

Learning Non-Flat HRMs. Figure 3 shows the LHRM learning curves for CRAFTWORLD (FRL) and WATERWORLD (WD). These settings are the most challenging due to the inclusion of dead-ends since (i) they hinder the observation of goal examples in level 1 tasks using random walks, (ii) the RMs must include rejecting states, (iii) formula options must avoid dead-ends, and (iv) call options must avoid invoking options leading to rejecting states. The curriculum method is visible: LHRM does not start learning anything in a task of a given level until tasks in previous levels are mastered. The convergence for high-level tasks is often fast due to the reuse of lower level HRMs and policies.

The average time (in seconds) spent exclusively on learning *all* HRMs is 1009.8 (122.3) for OP, 1622.6 (328.7) for OPL, 1031.6 (150.3) for FR, 1476.8 (175.3) for FRL, 35.4 (2.0) for WOD, and

67.0 (6.2) for WD. Including dead-ends (OPL, FRL, WD) incurs longer executions since (i) there is one more proposition, (ii) there are edges to the rejecting state(s), and (iii) there are dead-end traces to cover. We observe that the complexity of learning an HRM does not necessarily correspond with the task complexity (e.g., the times for OP and FRL are similar). Learning in WATERWORLD is faster than in CRAFTWORLD since the RMs have fewer states and there are fewer callable RMs.

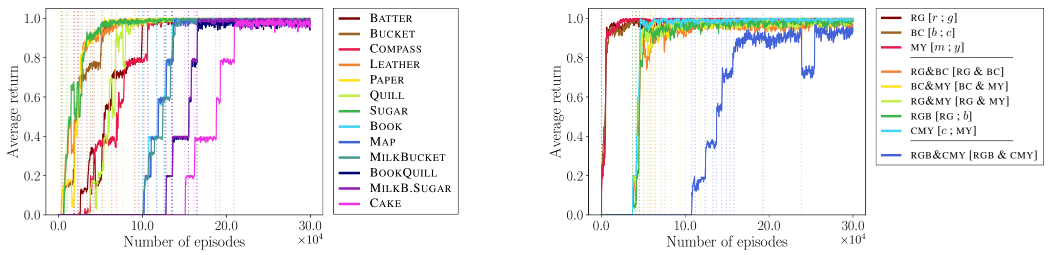


Figure 3: LHRM learning curves for CRAFTWORLD (FRL) and WATERWORLD (WD). The legend in WATERWORLD separates tasks by level, and the subtask order (in brackets) follows that introduced in Table 1. The dotted vertical lines correspond to episodes in which an HRM is learned.

By *restricting* the callable RMs to those required by the HRM (e.g., using just PAPER and LEATHER RMs to learn BOOK's HRM), there are fewer ways to label the edges of the induced root. We observe that learning is 5-7× faster using 20% less calls to the learner (i.e., less examples are needed) in CRAFTWORLD, and 1.5× faster in WATERWORLD. The number of callable RMs has a clear impact on HRM learning, becoming less scalable as the number of tasks and levels grows. Refining the set of callable RMs 'a priori' to speed up HRM learning is a direction for future work.

We evaluate the performance of *exploration with options* using the number of episodes needed to collect the ρ goal traces for a given task since the activation of its level. Intuitively, the agent will rarely move far from a region of the state space using primitive actions only, thus taking more time to collect the traces; in contrast, options enable the agent to explore the state space more efficiently. In the FRL setting of CRAFTWORLD, we observe that using primitive actions requires 128.1× more episodes than options in MILKBUCKET, the only level 2 task for which ρ traces are collected (although in just 2/5 runs). Likewise, primitive actions take 20.8× and 7.7× more episodes in OPL and WD respectively. In OP and WOD options are not beneficial since episodes are relatively long (1,000 steps), there are no dead-ends and it is easy to observe the different propositions.

Finally, we observe that using a single goal trace to learn the first HRM for each task ($\rho = \rho_s = 1$) instead of a set incurs timeouts across all CRAFTWORLD settings, thus empirically showing the usefulness of using several short traces to learn the HRMs.

Learning Flat HRMs. Learning a flat HRM is often less scalable than learning a non-flat equivalent since (i) previously learned HRMs cannot be reused, and (ii) the HRM may have exponentially more states and edges (see Theorem 2). We compare the performance of learning a non-flat HRM using LHRM with that of an equivalent flat HRM using LHRM, DeepSynth [21], LRM [50] and JIRP [55]. Akin to LHRM, the RMs induced with JIRP have explicit accepting states, whereas DeepSynth and LRM do not. We use OP and WOD instances for CRAFTWORLD and WATERWORLD respectively.

The non-flat HRM for MILKBUCKET is learned in 1.5 (0.2) seconds, whereas the flat HRMs take longer to learn: 3.2 (0.6) w/LHRM, 325.6 (29.7) w/DeepSynth, 347.5 (64.5) w/LRM and 17.1 (5.5) w/JIRP. LHRM and JIRP learn minimal RMs, hence producing the same RM consisting of 4 states and 3 edges. DeepSynth and LRM do not learn a minimal RM but one that is good at predicting the next possible label given the current one. In domains like ours where propositions can be observed anytime (i.e., without temporal dependencies between them), these methods tend to 'overfit' the input traces and produce large outputs that barely reflect the task's structure, e.g. DeepSynth learns RMs with 13.4 (0.4) states and 93.2 (1.7) edges. In contrast, methods learning minimal machines exclusively from observable traces may suffer from *overgeneralization* [3] in other domains (e.g., with temporally-dependent propositions). In more complex tasks such as BOOK, LHRM learns the non-flat HRM (see Figure 1b) in 191.2 (36.4) seconds, whereas methods learning the flat HRM (see Figure 1c) usually time out or, in the case of DeepSynth, learn bigger representations.

The performance of DeepSynth, LRM and JIRP is poor in WATERWORLD since they all learn RMs whose edges are labeled with proposition sets instead of formulas, unlike LHRM; thus, the RMs may require exponentially more edges, motivating the use of formulas for abstraction. For instance, the flat HRM for RG requires 64 edges instead of 2, and only LHRM and JIRP can learn it on time. All flat HRM learners time out in RG&BC, whereas the non-flat HRM is learned in 4.5 (0.3) seconds.

7 Related Work

Reward Machines and Composability. We define RMs that differ from the original ones [49, 51] in that (i) an RM can call other RMs, (ii) there are explicit accepting and rejecting states [17, 55], and (iii) transitions are labeled with propositional logic formulas instead of proposition sets [17]. Recent works *derive* RMs (and similar machines) from formal language specifications [4, 9] and expert demonstrations [10], or *learn* them from experience using discrete optimization [50], SAT solving [55], active learning [18, 56], state-merging [18, 54], program synthesis [21] or inductive logic programming [17]. Prior to our work, other ways of composing RMs have been considered. De Giacomo et al. [14] merge the state and reward transition functions of two RMs. Neary et al. [40] encode a multi-agent task using an RM, decompose it into one RM per agent and execute them in parallel. Alternative decompositions include subtask sequences called sketches [2], which are less expressive than RMs (the subtasks are run in a single order) and are not hierarchically composable. Context-free grammars defining a subset of English [12], formal languages [26, 33, 53] and logic-based algebras [39] have been also used to model task composability.

Hierarchical RL. Our method for exploiting HRMs resembles a hierarchy of DQNs [28]. Akin to option discovery methods, LHRM does not use handcrafted options but induces a set of them from experience. LHRM requires a set of propositions and tasks, which (approximately) determine the number of options that will be discovered; likewise, some option discovery methods impose maximum number of discoverable options [5, 35]. LHRM requires each task to be solved at least once before learning an HRM (and, hence, options), which is shared by other methods [37, 45]. The problem of discovering and using options for exploration has been considered before [7, 13, 25, 35]. While our options are not explicitly discovered for exploration, we leverage them to drive the search for goal traces in high-level tasks. Levy et al. [34] propose a method for learning policies from multiple hierarchical levels in parallel by training each level as if the lower levels were already optimal; similarly, we train call option policies exclusively using experiences where the invoked options achieve their goal. Besides, the hindsight action transitions resemble our mechanism for forming options based on transitions taken in hindsight.

Our method is close to the hierarchical abstract machines [HAMs, 41] HRL formalism due to the use of finite-state machines and the fact that one machine can call another (just like one option can invoke another option, as described in Section 4). HAMs, however, differ from HRMs (and RMs more generally) in that they are not defined over a set of propositions and they are non-deterministic. Programmable HAMs [1] extend HAMs by including parameterized subroutines, temporary interrupts, aborts and memory variables. To a limited extent, call contexts resemble parameterized subroutines.

Curriculum Learning. The method in LHRM is based on that by Pierrot et al. [42], who learn hierarchies of neural programs given the level of each program, which is similar to the height of an HRM. Andreas et al. [2] employ a similar method to prioritize tasks consisting of less high-level steps. The so-called ‘online’ method by Matiisen et al. [36] also keeps an estimate of the average return of the different tasks, but it is not presented in an HRL scenario. Wang et al. [53] introduce a method that initially learns linear temporal logic formulas for simple tasks, and progressively switches to harder ones leveraging previously learned formulas using a set of predefined templates. Deriving HRMs from (learned) templates is an interesting direction for future work.

8 Conclusions

We have here proposed (1) HRMs, a formalism that composes RMs in a hierarchy by allowing them to call each other, (2) an HRL method that exploits the structure of an HRM, and (3) a curriculum-based method for learning a collection of HRMs from traces. We have shown that non-flat HRMs have advantages with respect to their flat equivalents theoretically and empirically. We have demonstrated that learning a non-flat HRM is more scalable than learning an equivalent flat HRM since the constituent RMs are smaller and reusable.

LHRM assumes that the proposition set is known, dead-end indicators are shared across tasks, there is a fixed set of tasks and the height for each HRM is provided. Relaxing these *assumptions* by forming the propositions from raw data, conditioning policies to dead-ends, and letting the agent propose its own composable tasks are promising directions for future work. Other interesting extensions include *non-episodic* settings and methods for learning a *globally optimal* policies over HRMs.

References

[1] D. Andre and S. J. Russell. Programmable Reinforcement Learning Agents. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 1019–1025, 2000.

[2] J. Andreas, D. Klein, and S. Levine. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 166–175, 2017.

[3] D. Angluin. Inductive Inference of Formal Languages from Positive Data. *Inf. Control.*, 45(2): 117–135, 1980.

[4] B. Araki, X. Li, K. Vodrahalli, J. A. DeCastro, M. J. Fry, and D. Rus. The Logical Options Framework. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 307–317, 2021.

[5] P. Bacon, J. Harb, and D. Precup. The Option-Critic Architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1726–1734, 2017.

[6] A. G. Barto and S. Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[7] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 1471–1479, 2016.

[8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 41–48, 2009.

[9] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 6065–6073, 2019.

[10] A. Camacho, J. Varley, A. Zeng, D. Jain, A. Iscen, and D. Kalashnikov. Reward Machines for Vision-Based Robotic Manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 14284–14290, 2021.

[11] M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic Gridworld Environment for OpenAI Gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[12] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[13] W. Dabney, G. Ostrovski, and A. Barreto. Temporally-Extended ϵ-Greedy Exploration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[14] G. De Giacomo, M. Favorito, L. Iocchi, F. Patrizi, and A. Ronca. Temporal Logic Monitoring Rewards via Transducers. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 860–870, 2020.

[15] T. G. Dietterich, P. M. Domingos, L. Getoor, S. Muggleton, and P. Tadepalli. Structured machine learning: the next ten years. *Mach. Learn.*, 73(1):3–23, 2008.

[16] T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323, 1995.

[17] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. Induction and Exploitation of Subgoal Automata for Reinforcement Learning. *J. Artif. Intell. Res.*, 70:1031–1116, 2021.

[18] M. Gaon and R. I. Brafman. Reinforcement Learning with Non-Markovian Rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 3980–3987, 2020.

[19] M. Gelfond and Y. Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.

[20] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proceedings of the International Conference and Symposium on Logic Programming*, pages 1070–1080, 1988.

[21] M. Hasanbeig, N. Y. Jeppu, A. Abate, T. Melham, and D. Kroening. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 7647–7656, 2021.

[22] G. Hinton, N. Srivastava, and K. Swersky. Neural Networks for Machine Learning - Lecture 6a: Overview of Mini-Batch Gradient Descent. `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2012.

[23] M. Igl, K. Ciosek, Y. Li, S. Tschiatschek, C. Zhang, S. Devlin, and K. Hofmann. Generalization in Reinforcement Learning with Selective Noise Injection and Information Bottleneck. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 13956–13968, 2019.

[24] M. Jiang, E. Grefenstette, and T. Rocktäschel. Prioritized Level Replay. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 4940–4950, 2021.

[25] Y. Jinnai, J. W. Park, D. Abel, and G. D. Konidaris. Discovering Options for Exploration by Minimizing Cover Time. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3130–3139, 2019.

[26] K. Jothimurugan, R. Alur, and O. Bastani. A Composable Specification Language for Reinforcement Learning Tasks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 13021–13030, 2019.

[27] A. Karpathy. REINFORCEjs: WaterWorld demo. `http://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html`, 2015.

[28] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 3675–3683, 2016.

[29] M. Law. *Inductive Learning of Answer Set Programs*. PhD thesis, Imperial College London, UK, 2018.

[30] M. Law, A. Russo, and K. Broda. The ILASP System for Learning Answer Set Programs, 2015. URL `https://www.doc.ic.ac.uk/~ml1909/ILASP`.

[31] M. Law, A. Russo, and K. Broda. Iterative Learning of Answer Set Programs from Context Dependent Examples. *Theory Pract. Log. Program.*, 16(5-6):834–848, 2016.

[32] M. Law, A. Russo, and K. Broda. The Meta-program Injection Feature in ILASP. Technical report, June 2018. URL `https://www.doc.ic.ac.uk/~ml1909/ILASP/inject.pdf`.

[33] B. G. León, M. Shanahan, and F. Belardinelli. Systematic Generalisation through Task Temporal Logic and Deep Reinforcement Learning. *CoRR*, abs/2006.08767, 2020.

[34] A. Levy, G. D. Konidaris, R. P. Jr., and K. Saenko. Learning Multi-Level Hierarchies with Hindsight. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[35] M. C. Machado, M. G. Bellemare, and M. H. Bowling. A Laplacian Framework for Option Discovery in Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2295–2304, 2017.

[36] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman. Teacher-Student Curriculum Learning. *IEEE Trans. Neural Networks Learn. Syst.*, 31(9):3732–3740, 2020.

[37] A. McGovern and A. G. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 361–368, 2001.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[39] G. Nangue Tasse, S. D. James, and B. Rosman. A Boolean Task Algebra for Reinforcement Learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 9497–9507, 2020.

[40] C. Neary, Z. Xu, B. Wu, and U. Topcu. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 934–942, 2021.

[41] R. Parr and S. J. Russell. Reinforcement Learning with Hierarchies of Machines. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 1043–1049, 1997.

[42] T. Pierrot, G. Ligner, S. E. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, and N. de Freitas. Learning Compositional Neural Programs with Recursive Tree Search and Planning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 14646–14656, 2019.

[43] S. Sidor. Reinforcement Learning with Natural Language Signals. Master's thesis, Massachusetts Institute of Technology, 2016.

[44] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997. ISBN 978-0-534-94728-6.

[45] M. Stolle and D. Precup. Learning Options in Reinforcement Learning. In *Proceedings of the International Symposium on Abstraction, Reformulation and Approximation (SARA)*, pages 212–223, 2002.

[46] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[47] R. S. Sutton, D. Precup, and S. P. Singh. Intra-Option Learning about Temporally Abstract Actions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 556–564, 1998.

[48] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[49] R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2112–2121, 2018.

[50] R. Toro Icarte, E. Waldie, T. Q. Klassen, R. A. Valenzano, M. P. Castro, and S. A. McIlraith. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 15497–15508, 2019.

[51] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *J. Artif. Intell. Res.*, 73:173–208, 2022.

[52] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2094–2100, 2016.

[53] G. Wang, C. Trimbach, J. K. Lee, M. K. Ho, and M. L. Littman. Teaching a Robot Tasks of Arbitrary Complexity via Human Feedback. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 649–657, 2020.

[54] 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. *CoRR*, abs/1909.05912, 2019.

[55] 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 (ICAPS)*, pages 590–598, 2020.

[56] Z. Xu, B. Wu, A. Ojha, D. Neider, and U. Topcu. Active Finite Reward Automaton Inference and Reinforcement Learning Using Queries and Counterexamples. In *Proceedings of the International Cross-Domain Conference for Machine Learning and Knowledge Extraction (CD-MAKE)*, pages 115–135, 2021.

Hierarchies of Reward Machines (Supplementary Material)

The material presented in the following appendices complements the content introduced in the main paper through proofs, experimental details, and examples.

A Broader Impact

The proposed formalism has the potential to increase the applicability of reward machines (RMs) to model the structure of reinforcement learning (RL) tasks by composing them hierarchically. These hierarchies of RMs (HRMs) can be handcrafted or learned from experience and, in either of these two cases, the designer or the learner can choose to reuse existing RMs and ensemble them together to form a new, more abstract RM. Crucially, this does not only prevent the designer/learner from deriving each RM from scratch every time, but also enables leveraging previously learned policies associated with RMs that are known to work. Furthermore, HRMs are highly interpretable since their structure clearly reveals the high-level subgoals that need to be accomplished to complete a subtask. We foresee that these ideas are applicable in contexts where people not familiar with machine learning need to understand what the agent is trying to do and identify which pieces of behavior need to change or be composed into higher-level behavior. Ethical consequences mainly derive from the applicability of RL itself, which can be potentially used for both ‘good’ and ‘bad’ causes.

B Hierarchy Traversal Example

The HRM in Figure 1b accepts trace $\lambda = \langle \{\text{🎋}\}, \{\text{⛩}\}, \{\}, \{\text{🐮}\}, \{\text{⛩}\}, \{\text{🏛}\} \rangle$, whose traversal is $\mathcal{H}(\lambda) = \langle v_0, v_1, v_2, v_3, v_4, v_5, v_6 \rangle$, where:

$$
\begin{aligned}
v_0 &= \langle \mathcal{A}_0, u_0^0, \top, [] \rangle,\\
v_1 &= \delta_{\mathcal{H}}(v_0, \{\text{🎋}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_0, u_0^0, \top, [] \rangle, \{\text{🎋}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_1, u_1^0, \neg\text{🐮}, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top \rangle] \rangle, \{\text{🎋}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_\top, u_\top^0, \neg\text{🐮} \land \text{🎋}, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top \rangle, \langle u_1^0, u_1^1, \mathcal{A}_1, \mathcal{A}_\top, \text{🎋}, \neg\text{🐮} \rangle] \rangle, \{\text{🎋}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_1, u_1^1, \top, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top \rangle] \rangle, \bot)\\
&= \langle \mathcal{A}_1, u_1^1, \top, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top \rangle] \rangle,\\
v_2 &= \delta_{\mathcal{H}}(v_1, \{\text{⛩}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_1, u_1^1, \top, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top \rangle] \rangle, \{\text{⛩}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_\top, u_\top^0, \text{⛩}, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top \rangle, \langle u_1^1, u_1^A, \mathcal{A}_1, \mathcal{A}_\top, \text{⛩}, \top \rangle] \rangle, \{\text{⛩}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_1, u_1^A, \top, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top \rangle] \rangle, \bot)\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_0, u_0^1, \top, [] \rangle, \bot)\\
&= \langle \mathcal{A}_0, u_0^1, \top, [] \rangle,\\
v_3 &= \delta_{\mathcal{H}}(v_2, \{\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_0, u_0^1, \top, [] \rangle, \{\})\\
&= \langle \mathcal{A}_0, u_0^1, \top, [] \rangle,\\
v_4 &= \delta_{\mathcal{H}}(v_3, \{\text{🐮}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_0, u_0^1, \top, [] \rangle, \{\text{🐮}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_2, u_2^0, \top, [\langle u_0^1, u_0^3, \mathcal{A}_0, \mathcal{A}_2, \top, \top \rangle] \rangle, \{\text{🐮}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_\top, u_\top^0, \text{🐮}, [\langle u_0^1, u_0^3, \mathcal{A}_0, \mathcal{A}_2, \top, \top \rangle, \langle u_2^0, u_2^1, \mathcal{A}_2, \mathcal{A}_\top, \text{🐮}, \top \rangle] \rangle, \{\text{🐮}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_2, u_2^1, \top, [\langle u_0^1, u_0^3, \mathcal{A}_0, \mathcal{A}_2, \top, \top \rangle] \rangle, \bot)\\
&= \langle \mathcal{A}_2, u_2^1, \top, [\langle u_0^1, u_0^3, \mathcal{A}_0, \mathcal{A}_2, \top, \top \rangle] \rangle,\\
v_5 &= \delta_{\mathcal{H}}(v_4, \{\text{⛩}\})\\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_2, u_2^1, \top, [\langle u_0^1, u_0^3, \mathcal{A}_0, \mathcal{A}_2, \top, \top \rangle] \rangle, \{\text{⛩}\})
\end{aligned}
$$

$$
\begin{aligned}
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_\top, u_\top^0, \Box, [\langle u_0^1, u_0^3, \mathcal{A}_0, \mathcal{A}_2, \top, \top\rangle, \langle u_2^1, u_2^A, \mathcal{A}_2, \mathcal{A}_\top, \Box, \top\rangle]\rangle, \{\Box\}) \\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_2, u_2^A, \top, [\langle u_0^1, u_0^3, \mathcal{A}_0, \mathcal{A}_2, \top, \top\rangle]\rangle, \bot) \\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_0, u_0^3, \top, []\rangle, \bot) \\
&= \langle \mathcal{A}_0, u_0^3, \top, []\rangle, \\
v_6 &= \delta_{\mathcal{H}}(v_5, \{\blacksquare\}) \\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_0, u_0^3, \top, []\rangle, \{\blacksquare\}) \\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_\top, u_\top^0, \blacksquare, [\langle u_0^3, u_0^A, \mathcal{A}_0, \mathcal{A}_\top, \blacksquare, \top\rangle]\rangle, \{\blacksquare\}) \\
&= \delta_{\mathcal{H}}(\langle \mathcal{A}_0, u_0^A, \top, []\rangle, \bot) \\
&= \langle \mathcal{A}_0, u_0^A, \top, []\rangle.
\end{aligned}
$$

Example 1 shows the traversal but omits the intermediate applications of the hierarchical transition function $\delta_{\mathcal{H}}$.

C Equivalence to Flat Hierarchies of Reward Machines

In this section, we prove the theorems introduced in Section 3 regarding the equivalence of an arbitrary HRM to a flat HRM.

C.1 Proof of Theorem 1

We formally show that any HRM can be transformed into an equivalent one consisting of a single non-leaf RM. The latter HRM type is called *flat* since there is a single hierarchy level.

Definition 6. Given an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$, a constituent RM $\mathcal{A}_i \in \mathcal{A}$ is *flat* if its height h_i is 1.

Definition 7. An HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$ is *flat* if the root RM $\mathcal{A}_r$ is flat.

We now define what it means for two HRMs to be equivalent. This definition is based on that used in automaton theory [44].

Definition 8. Given a set of propositions $\mathcal{P}$ and a labeling function $\mathcal{L}$, two HRMs $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$ and $\mathcal{H}' = \langle \mathcal{A}', \mathcal{A}'_r, \mathcal{P}\rangle$ are *equivalent* if for any label trace λ one of the following conditions holds: (i) both HRMs accept λ, (ii) both HRMs reject λ, or (iii) neither of the HRMs accepts or rejects λ.

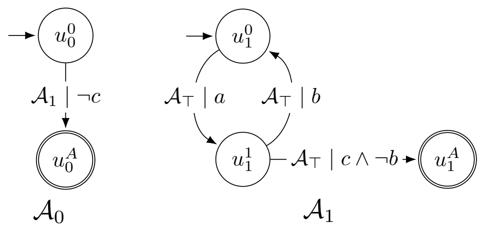
We now have all the required definitions to prove Theorem 1, which is restated below.

Theorem 1. *Given an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$, there exists an equivalent flat HRM $\bar{\mathcal{H}} = \langle \bar{\mathcal{A}}, \bar{\mathcal{A}}_r, \mathcal{P}\rangle$.*

To prove the theorem, we introduce an algorithm for flattening any HRM. Without loss of generality, we work on the case of an HRM with two hierarchy levels; that is, an HRM consisting of a root RM that calls flat RMs. Note that an HRM with an arbitrary number of levels can be flattened by considering the RMs in two levels at a time. We start flattening RMs in the second level (i.e., with height 2), which use RMs in the first level (by definition, these are already flat), and once the second level RMs are flat, we repeat the process with the levels above until the root is reached. This process is applicable since, by assumption, the hierarchies do not have cyclic dependencies nor recursion. For simplicity, we use the MDP reward assumption made in Section 2, i.e. the reward transition function of any RM $\mathcal{A}_i$ is $r_i(u, u') = \mathbb{1}[u \notin U_i^A \wedge u' \in U_i^A]$ like in Section 4. However, the proof below could be adapted to arbitrary definitions of $r_i(u, u')$.

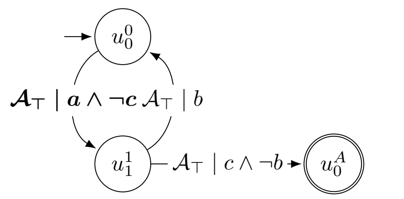
Preliminary Transformation Algorithm. Before proving Theorem 1, we introduce an intermediate step that transforms a flat HRM into an equivalent one that takes contexts with which it may be called into account. Remember that a call to an RM is associated with a context. In the case of two-level HRMs such as the ones we are considering in this flattening process, the context and the exit condition from the called flat RM must be satisfied. Crucially, the context must only be satisfied at the time of the call; that is, it only lasts for a single transition. Therefore, if we revisit the initial state of the called RM by taking an edge to it, the context should not be checked anymore.

To make the need for this transformation clearer, we use the HRM illustrated in Figure 4a. The flattening algorithm described later embeds the called RM into the caller one; crucially, the context of the call is taken into account by putting it in conjunction with the outgoing edges from the initial state of the called RM.[2] Figure 4b is a flat HRM obtained using the flattening algorithm; however, it does not behave like the HRM in Figure 4a. Following the definition of the hierarchical transition function $\delta_{\mathcal{H}}$, the context of a call only lasts for a single transition in the called RM in Figure 4a (i.e., $a \wedge \neg c$ is only checked when $\mathcal{A}_1$ is started), but the context is kept permanently in Figure 4b, which is problematic if we go back to the initial state at some point. We later come back to this example after presenting the transformation algorithm.



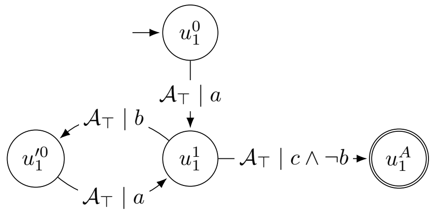
u_0^0
$\mathcal{A}_1 \mid \neg c$
u_0^A
$\mathcal{A}_0$
u_1^0
$\mathcal{A}_\top \mid a$
$\mathcal{A}_\top \mid b$
u_1^1
$\mathcal{A}_\top \mid c \wedge \neg b$
u_1^A
$\mathcal{A}_1$

(a) Original HRM with root $\mathcal{A}_0$.



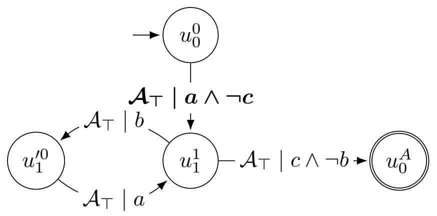
u_0^0
$\boldsymbol{\mathcal{A}_\top \mid a \wedge \neg c}$
$\mathcal{A}_\top \mid b$
u_1^1
$\mathcal{A}_\top \mid c \wedge \neg b$
u_0^A

(b) Flattened HRM without transforming $\mathcal{A}_1$.



u_1^0
$\mathcal{A}_\top \mid a$
$\mathcal{A}_\top \mid b$
$u_1'^0$
u_1^1
$\mathcal{A}_\top \mid c \wedge \neg b$
u_1^A
$\mathcal{A}_\top \mid a$

(c) Transformed $\mathcal{A}_1$ from (a).



u_0^0
$\boldsymbol{\mathcal{A}_\top \mid a \wedge \neg c}$
$\mathcal{A}_\top \mid b$
$u_1'^0$
u_1^1
$\mathcal{A}_\top \mid c \wedge \neg b$
u_0^A
$\mathcal{A}_\top \mid a$

(d) Flattened HRM after transforming $\mathcal{A}_1$.

Figure 4: Example to justify the need for the preliminary transformation algorithm.

To deal with the situation above, we need to transform an RM to ensure that contexts are only checked once from the initial state. We describe this transformation as follows. Given a flat HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$ with root $\mathcal{A}_r = \langle U_r, \mathcal{P}, \varphi_r, r_r, u_r^0, U_r^A, U_r^R \rangle$, we construct a new HRM $\mathcal{H}' = \langle \mathcal{A}', \mathcal{A}_r', \mathcal{P} \rangle$ with root $\mathcal{A}_r' = \langle U_r', \mathcal{P}, \varphi_r', r_r', u_r^0, U_r^A, U_r^R \rangle$ such that:

- $U_r' = U_r \cup \{\hat{u}_r^0\}$, where $\hat{u}_r^0$ plays the role of the initial state after the first transition is taken.
- The state transition function φ_r' is built by copying φ_r and applying the following:
  1. Remove the edges to the actual initial state from any state $v \in U_r'$: $\varphi_r'(v, u_r^0, \mathcal{A}_\top) = \bot$. Note that since the RM is flat, the only callable RM is the leaf $\mathcal{A}_\top$.
  2. Add edges to the dummy initial state $\hat{u}_r^0$ from all states $v \in U_r'$ that had an edge to the actual initial state: $\varphi_r'(v, \hat{u}_r^0, \mathcal{A}_\top) = \varphi_r(v, u_r^0, \mathcal{A}_\top)$.
  3. Add edges from the dummy initial state $\hat{u}_r^0$ to all those states $v \in U_r'$ that the actual initial state u_r^0 points to: $\varphi_r'(\hat{u}_r^0, v, \mathcal{A}_\top) = \varphi_r'(u_r^0, v, \mathcal{A}_\top)$.
- The reward transition function $r_r'(u, u') = \mathbb{1}[u \notin U_r^A \wedge u' \in U_r^A]$ is defined as stated at the beginning of the section.

The HRM $\mathcal{H}'$ is such that $\mathcal{A}' = \{\mathcal{A}_r', \mathcal{A}_\top\}$. Note that this transformation is only required in HRMs where the RMs have initial states with incoming edges.

We now prove that this transformation is correct; that is, the HRMs are equivalent. There are two cases depending on whether the initial state has incoming edges or not. First, if the initial state u_r^0 does not have incoming edges, step 1 does not remove any edges going to u_r^0, and step 2 does not add any edges going to $\hat{u}_r^0$, making it unreachable. Even though edges from $\hat{u}_r^0$ to other states may be added, it is irrelevant since it is unreachable. Therefore, we can safely say that in this case, the transformed HRM is equivalent to the original one. Second, if the initial state has incoming edges, we prove equivalence by examining the traversals $\mathcal{H}(\lambda)$ and $\mathcal{H}'(\lambda)$ for the original HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$ and the transformed one $\mathcal{H}' = \langle \mathcal{A}', \mathcal{A}_r', \mathcal{P} \rangle$ given a generic label trace $\lambda = \langle L_0, \ldots, L_n \rangle$. By construction, both $\mathcal{H}(\lambda)$ and $\mathcal{H}'(\lambda)$ will be identical until reaching a state w with an outgoing transition to u_r^0 in the case of $\mathcal{H}$ and the dummy initial state $\hat{u}_r^0$ in the case of $\mathcal{H}'$.

[2]We refer the reader to the 'Flattening Algorithm' description introduced later for specific details.

More specifically, upon reaching w and satisfying an outgoing formula to the aforementioned states, the traversals are:

$$\mathcal{H}(\lambda) = \langle\langle \mathcal{A}_r, u_r^0, \top, []\rangle, \dots, \langle \mathcal{A}_r, w, \top, []\rangle\rangle,$$
$$\mathcal{H}'(\lambda) = \langle\langle \mathcal{A}'_r, u_r^0, \top, []\rangle, \dots, \langle \mathcal{A}'_r, w, \top, []\rangle\rangle.$$

By construction, state w is in both HRMs, and both of the aforementioned transitions from this state are associated with the same formula (i.e., $\varphi_r(w, u_r^0, \mathcal{A}_\top) = \varphi'_r(w, \hat{u}_r^0, \mathcal{A}_\top)$). Therefore, if one of them is satisfied, the other will be too, and the traversals will become:

$$\mathcal{H}(\lambda) = \langle\langle \mathcal{A}_r, u_r^0, \top, []\rangle, \dots, \langle \mathcal{A}_r, w, \top, []\rangle, \langle \mathcal{A}_r, u_r^0, \top, []\rangle\rangle,$$
$$\mathcal{H}'(\lambda) = \langle\langle \mathcal{A}'_r, u_r^0, \top, []\rangle, \dots, \langle \mathcal{A}'_r, w, \top, []\rangle, \langle \mathcal{A}'_r, \hat{u}_r^0, \top, []\rangle\rangle.$$

We stay in u_r^0 and $\hat{u}_r^0$ until a transition to a state w' is satisfied. By construction, w' is in both HRMs and the same formula is satisfied (i.e., $\varphi_r(u_r^0, w', \mathcal{A}_\top) = \varphi'_r(\hat{u}^0, w', \mathcal{A}_\top)$). The hierarchy traversals then become:

$$\mathcal{H}(\lambda) = \langle\langle \mathcal{A}_r, u_r^0, \top, []\rangle, \dots, \langle \mathcal{A}_r, w, \top, []\rangle, \langle \mathcal{A}_r, u_r^0, \top, []\rangle, \dots, \langle \mathcal{A}_r, u_r^0, \top, []\rangle, \langle \mathcal{A}_r, w', \top, []\rangle\rangle,$$
$$\mathcal{H}'(\lambda) = \langle\langle \mathcal{A}'_r, u_r^0, \top, []\rangle, \dots, \langle \mathcal{A}'_r, w, \top, []\rangle, \langle \mathcal{A}'_r, \hat{u}_r^0, \top, []\rangle, \dots, \langle \mathcal{A}'_r, \hat{u}_r^0, \top, []\rangle, \langle \mathcal{A}'_r, w', \top, []\rangle\rangle.$$

From here both traversals will be the same until transitions to u_r^0 and $\hat{u}_r^0$ are respectively satisfied again (if any) in $\mathcal{H}$ and $\mathcal{H}'$. Clearly, the only change in $\mathcal{H}(\lambda)$ with respect to $\mathcal{H}'(\lambda)$ (except for the different roots) is that the hierarchy states of the form $\langle \mathcal{A}'_r, \hat{u}_r^0, \top, []\rangle$ in the latter appear as $\langle \mathcal{A}_r, u_r^0, \top, []\rangle$ in the former. We now check if the equivalence conditions from Definition 8 hold:

- If $\mathcal{H}(\lambda)$ ends with state u_r^0, $\mathcal{H}'(\lambda)$ ends with state $\hat{u}_r^0$ following the reasoning above. By construction, neither of these states is accepting or rejecting; therefore, neither of these HRMs accepts or rejects λ.
- If $\mathcal{H}(\lambda)$ ends with state w, $\mathcal{H}'(\lambda)$ will also end with this state following the reasoning above. Therefore, if w is an accepting state, both HRMs accept λ; if w is a rejecting state, both HRMs reject λ; and if w is not an accepting or rejecting state, neither of the HRMs accepts or rejects λ.

Since all equivalence conditions are satisfied for any trace λ, $\mathcal{H}$ and $\mathcal{H}'$ are equivalent.

Figure 4c exemplifies the output of the transformation algorithm given $\mathcal{A}_1$ in Figure 4a as input, whereas Figure 4d is the output of the flattening algorithm discussed next, which properly handles the context unlike the HRM in Figure 4b.

Flattening Algorithm. We describe the algorithm for flattening an HRM. As previously stated, we assume without loss of generality that the HRM to be flattened consists of two hierarchy levels (i.e., the root calls flat RMs). We also assume that the flat RMs have the form produced by the previously presented transformation algorithm.

Given an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$ with root $\mathcal{A}_r = \langle U_r, \mathcal{P}, \varphi_r, r_r, u_r^0, U_r^A, U_r^R\rangle$, we build a flat RM $\bar{\mathcal{A}}_r = \langle \bar{U}_r, \mathcal{P}, \bar{\varphi}_r, \bar{r}_r, \bar{u}_r^0, \bar{U}_r^A, \bar{U}_r^R\rangle$ using the following steps:

1. Copy the sets of states and initial state from $\mathcal{A}_r$ (i.e., $\bar{U}_r = U_r, \bar{u}_r^0 = u_r^0, \bar{U}_r^A = U_r^A, \bar{U}_r^R = U_r^R$).
2. Loop through the non-false entries of the transition function φ_r and decide what to copy. That is, for each triplet $(u, u', \mathcal{A}_j)$ where $u, u' \in U_r$ and $\mathcal{A}_j \in \mathcal{A}$ such that $\varphi_r(u, u', \mathcal{A}_j) \neq \bot$:
   (a) If $\mathcal{A}_j = \mathcal{A}_\top$ (i.e., the called RM is the leaf), we copy the transition: $\bar{\varphi}_r(u, u', \mathcal{A}_\top) = \varphi_r(u, u', \mathcal{A}_\top)$.
   (b) If $\mathcal{A}_j \neq \mathcal{A}_\top$, we embed the transition function of $\mathcal{A}_j = \langle U_j, \mathcal{P}, \varphi_j, r_j, u_j^0, U_j^A, U_j^R\rangle$ into $\bar{\mathcal{A}}_r$. Remember that $\mathcal{A}_j$ is flat. To do so, we run the following steps:
      i. Update the set of states by adding all non-initial and non-accepting states from $\mathcal{A}_j$. Similarly, the set of rejecting states is also updated by adding all rejecting states of the called RM. The initial and accepting states from $\mathcal{A}_j$ are unimportant: their roles are played by u and u' respectively. In contrast, the rejecting states are important since, by assumption, they are global. Note that the added states v are renamed to

$v_{u,u',j}$ in order to take into account the edge being embedded: if the same state v was reused for another edge, then we would not be able to distinguish them.

$$\bar{U}_r = \bar{U}_r \cup \left\{ v_{u,u',j} \mid v \in \left(U_j \setminus \left(\{u_j^0\} \cup U_j^A \right) \right) \right\},$$
$$\bar{U}_r^R = \bar{U}_r^R \cup \left\{ v_{u,u',j} \mid v \in U_j^R \right\}.$$

ii. Embed the transition function φ_j of $\mathcal{A}_j$ into $\bar{\varphi}_r$. Since $\mathcal{A}_j$ is flat, we can make copies of the transitions straightaway: the only important thing is to check whether these transitions involve initial or accepting states which, as stated before, are going to be replaced by u and u' accordingly. Given a triplet $(v, w, \mathcal{A}_\top)$ such that $v, w \in U_j$ and for which $\varphi_j(v, w, \mathcal{A}_\top) = \phi$ and $\phi \neq \perp$ we update $\bar{\varphi}_r$ as follows:[3]

A. If $v = u_j^0$ and $w \notin U_j^A$, then $\bar{\varphi}_r(u, w_{u,u',j}, \mathcal{A}_\top) = \text{DNF}(\phi \wedge \varphi_r(u, u', \mathcal{A}_j))$. The initial state of $\mathcal{A}_j$ has been substituted by u, we use the clone of w associated with the call $(w_{u,u',j})$, and append the context of the call to $\mathcal{A}_j$ to the formula ϕ.

B. If $v = u_j^0$ and $w \in U_j^A$, then $\bar{\varphi}_r(u, u', \mathcal{A}_\top) = \text{DNF}(\phi \wedge \varphi_r(u, u', \mathcal{A}_j))$. Like the previous case but performing two substitutions: u replaces v and u' replaces w. The context is appended since it is a transition from the initial state of $\mathcal{A}_j$.

C. If $v \neq u_j^0$ and $w \in U_j^A$, then $\bar{\varphi}_r(v_{u,u',j}, u', \mathcal{A}_\top) = \phi$. We substitute the accepting state w by u', and use the clone of v associated with the call $(v_{u,u',j})$. This time the call's context is not added since v is not the initial state of $\mathcal{A}_j$.

D. If none of the previous cases holds, there are no substitutions to be made nor contexts to be taken into account. Hence, $\bar{\varphi}_r(v_{u,u',j}, w_{u,u',j}, \mathcal{A}_\top) = \phi$. We just use the clones of v and w corresponding to the call ($v_{u,u',j}$ and $w_{u,u',j}$).

3. We apply the transformation algorithm we described before, and form a new flat HRM $\bar{\mathcal{H}} = \langle \{\bar{\mathcal{A}}_r, \mathcal{A}_\top\}, \bar{\mathcal{A}}_r, \mathcal{P} \rangle$ with the flattened (and transformed) root $\bar{\mathcal{A}}_r$.

The reward transition function $r_r'(u, u') = \mathbb{1}[u \notin \bar{U}_r^A \wedge u' \in \bar{U}_r^A]$ is defined as stated at the beginning of the section. Note that u might not necessarily be a state of the non-flat root, but derived from an RM with lower height.

We now have everything to prove the previous theorem. Without loss of generality and for simplicity, we assume that the transformation algorithm has not been applied over the flattened root (we have already shown that the transformation already produces an equivalent flat machine).

Theorem 1. *Given an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$, there exists an equivalent flat HRM $\bar{\mathcal{H}} = \langle \bar{\mathcal{A}}, \bar{\mathcal{A}}_r, \mathcal{P} \rangle$.*

Proof. Let us assume that an HRM $\bar{\mathcal{H}} = \langle \bar{\mathcal{A}}, \bar{\mathcal{A}}_r, \bar{\delta}_{\mathcal{H}} \rangle$, where $\bar{\mathcal{A}}_r = \langle \bar{U}_r, \mathcal{P}, \bar{\varphi}_r, \bar{r}_r, \bar{u}_r^0, \bar{U}_r^A, \bar{U}_r^R \rangle$, is a flat HRM that results from applying the flattening algorithm on an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$, where $\mathcal{A}_r = \langle U_r, \mathcal{P}, \varphi_r, r_r, u_r^0, U_r^A, U_r^R \rangle$. For these HRMs to be equivalent, any label trace $\lambda = \langle L_0, \ldots, L_n \rangle$ must satisfy one of the conditions in Definition 8. To prove the equivalence, we examine the hierarchy traversals $\mathcal{H}(\lambda)$ and $\bar{\mathcal{H}}(\lambda)$ given a generic label trace λ.

Let $u \in U_r$ be a state in the root $\mathcal{A}_r$ of $\mathcal{H}$ and let $\varphi_r(u, u', \mathcal{A}_\top)$ be a satisfied transition from that state. By construction, u is also in the root $\bar{\mathcal{A}}_r$ of the flat hierarchy $\bar{\mathcal{H}}$, and $\bar{\mathcal{A}}_r$ has an identical transition $\bar{\varphi}_r(u, u', \mathcal{A}_\top)$, which must also be satisfied. If the hierarchy states are $\langle \mathcal{A}_r, u, \top, [] \rangle$ and $\langle \bar{\mathcal{A}}_r, u, \top, [] \rangle$ for $\mathcal{H}$ and $\bar{\mathcal{H}}$ respectively, then the next hierarchy states upon application of $\delta_{\mathcal{H}}$ will be $\langle \mathcal{A}_r, u', \top, [] \rangle$ and $\langle \bar{\mathcal{A}}_r, u', \top, [] \rangle$. Therefore, both HRMs behave equivalently when calls to the leaf RM are made.

We now examine what occurs when a non-leaf RM is called in $\mathcal{H}$. Let $\varphi_r(u, u', \mathcal{A}_j)$ be a satisfied transition in $\mathcal{A}_r$, and let $\varphi_j(u_j^0, w, \mathcal{A}_\top)$ be a satisfied transition from $\mathcal{A}_j$'s initial state. By construction, $\bar{\mathcal{A}}_r$ contains a transition whose associated formula is the conjunction of the previous two, i.e. $\varphi_r(u, u', \mathcal{A}_j) \wedge \varphi_j(u_j^0, w, \mathcal{A}_\top)$. Now, the hierarchy traversals will be different depending on w:

- If $w \notin U_j^A$ (i.e., w is not an accepting state of $\mathcal{A}_j$), by construction, $\bar{\mathcal{A}}_r$ contains the transition $\bar{\varphi}_r(u, w_{u,u',j}, \mathcal{A}_\top) = \varphi_r(u, u', \mathcal{A}_j) \wedge \varphi_j(u_j^0, w, \mathcal{A}_\top)$. If the current hierarchy

[3]We do not to cover the case where v is an accepting state since, by assumption, there are no outgoing transitions from it. In the case of rejecting states, we keep all of them as explained in the previous case and, therefore, there are no substitutions to be made. We also do not cover the case where $w = u_j^0$ since the input flat machines never have edges to their initial states, but to the dummy initial state.

states are (the equivalent) $\langle \mathcal{A}_r, u, \top, [] \rangle$ and $\langle \bar{\mathcal{A}}_r, u, \top, [] \rangle$ for $\mathcal{H}$ and $\bar{\mathcal{H}}$, then the next hierarchy states upon application of $\delta_{\mathcal{H}}$ are $\langle \mathcal{A}_j, w, \top, [\langle u, u', \mathcal{A}_r, \mathcal{A}_j, \varphi_r(u, u', \mathcal{A}_j), \top \rangle] \rangle$ and $\langle \bar{\mathcal{A}}_r, w_{u,u',j}, \top, [] \rangle$. These hierarchy states are equivalent since $w_{u,u',j}$ is a clone of w that saves all the call information (i.e., a call to machine $\mathcal{A}_j$ for transitioning from u to u').

- If $w \in U_j^A$ (i.e., w is an accepting state of $\mathcal{A}_j$), by construction, $\bar{\mathcal{A}}_r$ contains the transition $\bar{\varphi}_r(u, u', \mathcal{A}_\top) = \varphi_r(u, u', \mathcal{A}_j) \wedge \varphi_j(u_j^0, w, \mathcal{A}_\top)$. If the current hierarchy states are (the equivalent) $\langle \mathcal{A}_r, u, \top, [] \rangle$ and $\langle \bar{\mathcal{A}}_r, u, \top, [] \rangle$ for $\mathcal{H}$ and $\bar{\mathcal{H}}$, then the next hierarchy states upon application of $\delta_{\mathcal{H}}$ are $\langle \mathcal{A}_r, u', \top, [] \rangle$ and $\langle \bar{\mathcal{A}}_r, u', \top, [] \rangle$. These hierarchy states are clearly equivalent since the machine states are exactly the same.

We now check the case in which we are inside a called RM. Let $\varphi_r(u, u', \mathcal{A}_j)$ be the transition that caused $\mathcal{H}$ to start running $\mathcal{A}_j$, and let $\varphi_j(v, w, \mathcal{A}_\top)$ be a satisfied transition within $\mathcal{A}_j$ such that $v \neq u_j^0$. By construction, $\bar{\mathcal{A}}_r$ contains a transition associated with the same formula $\varphi_j(v, w, \mathcal{A}_\top)$. The hierarchy traversals vary depending on w:

- If $w \notin U_j^A$ (i.e., w is not an accepting state of $\mathcal{A}_j$), by construction, $\bar{\mathcal{A}}_r$ contains the transition $\bar{\varphi}_r(v_{u,u',j}, w_{u,u',j}, \mathcal{A}_\top) = \varphi_j(v, w, \mathcal{A}_\top)$. For the transition to be taken in $\mathcal{H}$, the hierarchy state must be $\langle \mathcal{A}_j, v, \top, [\langle u, u', \mathcal{A}_r, \mathcal{A}_j, \varphi_r(u, u', \mathcal{A}_j), \top \rangle] \rangle$, whereas in $\bar{\mathcal{H}}$ it will be $\langle \bar{\mathcal{A}}_r, v_{u,u',j}, \top, [] \rangle$. These hierarchy states are clearly equivalent: $v_{u,u',j}$ is a clone of v that saves all information related to the call being made (the called machine, and the starting and resulting states in the transition). Upon application of $\delta_{\mathcal{H}}$, the hierarchy states will remain equivalent: $\langle \mathcal{A}_j, w, \top, [\langle u, u', \mathcal{A}_r, \mathcal{A}_j, \varphi_r(u, u', \mathcal{A}_j), \top \rangle] \rangle$ and $\langle \bar{\mathcal{A}}_r, w_{u,u',j}, \top, [] \rangle$ (again $w_{u,u',j}$ saves all the call information, just like the stack).
- If $w \in U_j^A$ (i.e., w is an accepting state of $\mathcal{A}_j$), by construction, $\bar{\mathcal{A}}_r$ contains the transition $\bar{\varphi}_r(v_{u,u',j}, u', \mathcal{A}_\top) = \varphi_j(v, w, \mathcal{A}_\top)$. This case corresponds to that where control is returned to the calling RM. Like in the previous case, for the transition to be taken in $\mathcal{H}$, the hierarchy state must be $\langle \mathcal{A}_j, v, \top, [\langle u, u', \mathcal{A}_r, \mathcal{A}_j, \varphi_r(u, u', \mathcal{A}_j), \top \rangle] \rangle$, whereas in $\bar{\mathcal{H}}$ it will be $\langle \bar{\mathcal{A}}_r, v_{u,u',j}, \top, [] \rangle$. The resulting hierarchy states then become $\langle \mathcal{A}_r, u', \top, [] \rangle$ and $\langle \bar{\mathcal{A}}_r, u', \top, [] \rangle$ respectively, which are clearly equivalent (the state is exactly the same and both come from equivalent hierarchy states).

We have shown both HRMs have equivalent traversals for any given trace, implying that both will accept, reject, or not accept nor reject a trace. Therefore, the HRMs are equivalent. $\square$

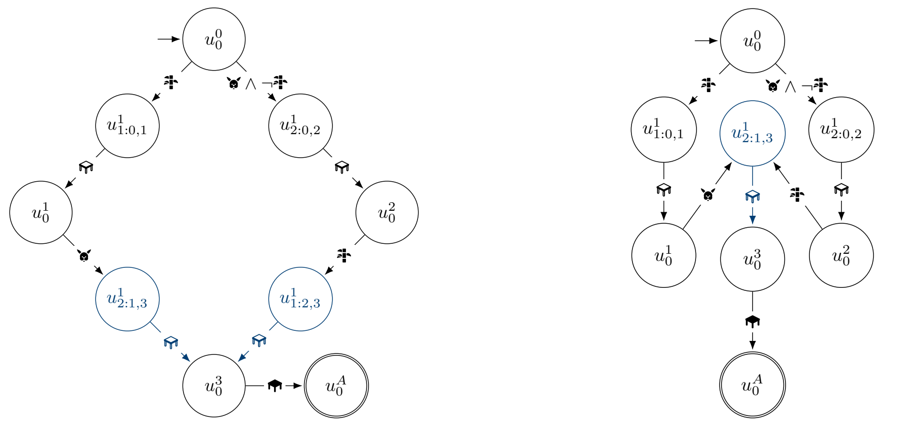
Figure 5a shows the result of applying the flattening algorithm on the BOOK HRM shown in Figure 1b. Note that the resulting HRM can be compressed: there are two states having an edge with the same label to a specific state. Indeed, the presented algorithm might not produce the smallest possible flat equivalent. Figure 5b shows the resulting compressed HRM, which is like Figure 1c but naming the states following the algorithm for clarity. Estimating how much a flat HRM (or any HRM) can be compressed and designing an algorithm to perform such compression are left as future work.

C.2 Proof of Theorem 2

We prove Theorem 2 by first characterizing an HRM $\mathcal{H}$ using a set of abstract parameters. Then, we describe how the number of states and edges in an HRM and its corresponding flat equivalent are computed, and use these quantities to give an example for which the theorem holds. The parameters are the following:

- The height of the root h_r.
- The number of RMs with height i, $N^{(i)}$.
- The number of states in RMs with height i, $U^{(i)}$.
- The number of edges from each state in RMs with height i, $E^{(i)}$.

We assume that (i) RMs with height i only call RMs with height $i-1$; (ii) all RMs have a single accepting state and no rejecting states; (iii) all RMs except for the root are called; and (iv) the HRM



(a) Without compression. (b) With compression.

Figure 5: Results of flattening the HRM in Figure 1b. The notation $u^i_{j:x,y}$ denotes the i-th state of RM j in the call between states x and y in the parent RM. Note that x and y appear only if that state comes from a called RM. The blue states and edges in (a) can be compressed as shown in (b).

is well-formed (i.e., it behaves deterministically and there are no cyclic dependencies). Note that $N^{(h_r)} = 1$ since there is a single root. Assumption (ii) is safe to make since a single accepting state is enough, and helps simplify the counting since only some RMs could have rejecting states. Assumption (iii) ensures that the flat HRM will comprise all RMs in the original HRM.

The *number of states* $|\mathcal{H}|$ *in the HRM* $\mathcal{H}$ is obtained by summing the number of states of each RM:

$$|\mathcal{H}| = \sum_{i=1}^{h_r} N^{(i)} U^{(i)}.$$

The *number of states* $|\bar{\mathcal{H}}|$ *in the flat HRM* $\bar{\mathcal{H}}$ is given by the number of states in the flattened root RM

$$|\bar{\mathcal{H}}| = \bar{U}^{(h_r)},$$

where $\bar{U}^{(i)}$ is the number of states in the flattened representation of an RM with height i, which is recursively defined as:

$$\bar{U}^{(i)} = \begin{cases} U^{(i)} & \text{if } i = 1, \\ U^{(i)} + \left(\bar{U}^{(i-1)} - 2\right)\left(U^{(i)} - 1\right) E^{(i)} & \text{if } i > 1. \end{cases}$$

That is, the number of states in a flattened RM with height i has all states that the non-flat HRM had. In addition, for each of the $U^{(i)} - 1$ non-accepting states in the non-flat RM, there are $E^{(i)}$ edges, each of which calls an RM with height $i - 1$ whose number of states is $\bar{U}^{(i-1)}$. These edges are replaced by the called RM except for the initial and accepting states, whose role is now played by the states involved in the substituted edge (hence the -2). This construction process corresponds to the one used to prove Theorem 1.

The *total of number of edges* in an HRM is given by:

$$\sum_{i=1}^{h_r} N^{(i)} (U^{(i)} - 1) E^{(i)},$$

where $(U^{(i)} - 1)E^{(i)}$ is the total number of edges in an RM with height i (the -1 is because the accepting state is discarded), so $N^{(i)}(U^{(i)} - 1)E^{(i)}$ determines how many edges there are across RMs with height i.

The *total number of edges in the flat HRM* is given by the total number of edges in the flattened root RM, $\bar{E}^{(h_r)}$, where $\bar{E}^{(i)}$ is the total number of edges in the flattened representation of an RM with height i, which is recursively defined as follows:

$$\bar{E}^{(i)} = \begin{cases} (U^{(i)} - 1)E^{(i)} & \text{if } i = 1, \\ (U^{(i)} - 1)E^{(i)} \bar{E}^{(i-1)} & \text{if } i > 1. \end{cases}$$

That is, each of the $(U^{(i)}-1)E^{(i)}$ edges in an RM with height i is replaced by $\bar{E}^{(i-1)}$ edges given by an RM with height $i-1$ (if any).

The *key intuition* is that an HRM with root height $h_r > 1$ is beneficial representation-wise if the number of calls across RMs with height i is higher than the number of RMs with height $i-1$; in other words, RMs with lower heights are being reused. Numerically, the total number of edges/calls in an RM with height i is $(U^{(i)}-1)E^{(i)}$ and, therefore, the total number of calls across RMs with height i is $(U^{(i)}-1)E^{(i)}N^{(i)}$. If this quantity is higher than $N^{(i-1)}$, then RMs with lower heights are reused, and therefore having RMs with different heights is beneficial.

Theorem 2. *Let $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$ be an HRM and let h_r be the height of its root $\mathcal{A}_r$. The number of states and edges in an equivalent flat HRM $\bar{\mathcal{H}}$ may be exponential in h_r.*

Proof. By example. Let $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$ be an HRM whose root $\mathcal{A}_r$ has height h_r and is parameterized by $N^{(i)} = 1$, $U^{(i)} = 3$, $E^{(i)} = 1$ for $i = 1, \ldots, h_r$. Figure 6 shows an instance of this hierarchy. Let us write the number of states in the flat RMs of each level:

$$\begin{aligned}
\bar{U}^{(1)} &= U^{(1)} = 3, \\
\bar{U}^{(2)} &= U^{(2)} + \left(\bar{U}^{(1)} - 2\right)\left(U^{(2)} - 1\right)E^{(2)} = 3 + (3-2)(3-1)1 = 5, \\
\bar{U}^{(3)} &= U^{(3)} + \left(\bar{U}^{(2)} - 2\right)\left(U^{(3)} - 1\right)E^{(3)} = 3 + (5-2)(3-1)1 = 9, \\
&\vdots \\
\bar{U}^{(i)} &= 2\bar{U}^{(i-1)} - 1 = 2^i + 1.
\end{aligned}$$

Hence, the number of states in the flat HRM is $|\bar{\mathcal{H}}| = \bar{U}^{(h_r)} = 2^{h_r} + 1$, showing that the number of states in the flat HRM grows exponentially with the height of the root. In contrast, the number of states in the HRM grows linearly with the height of the root, $|\mathcal{H}| = \sum_{i=1}^{h_r} N^{(i)}U^{(i)} = \sum_{i=1}^{h_r} 1 \cdot 3 = 3h_r$.

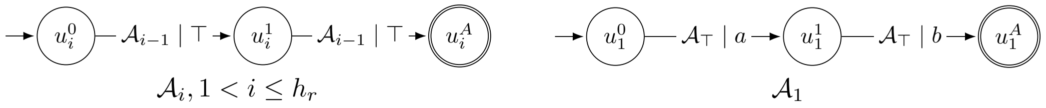

u_i^0
$\mathcal{A}_{i-1} \mid \top$
u_i^1
$\mathcal{A}_{i-1} \mid \top$
u_i^A
$\mathcal{A}_i, 1 < i \leq h_r$
u_1^0
$\mathcal{A}_\top \mid a$
u_1^1
$\mathcal{A}_\top \mid b$
u_1^A
$\mathcal{A}_1$

Figure 6: Example of an HRM whose root has height h_r used in the proof of Theorem 2.

In the case of the total number of edges, we again write some iterations to derive a general expression:

$$\begin{aligned}
\bar{E}^{(1)} &= (U^{(1)} - 1)E^{(1)} = (3-1)1 = 2, \\
\bar{E}^{(2)} &= (U^{(2)} - 1)E^{(2)}\bar{E}^{(1)} = (3-1) \cdot 1 \cdot 2 = 4, \\
\bar{E}^{(3)} &= (U^{(3)} - 1)E^{(3)}\bar{E}^{(2)} = (3-1) \cdot 1 \cdot 4 = 8, \\
&\vdots \\
\bar{E}^{(i)} &= 2\bar{E}^{(i-1)} = 2^i.
\end{aligned}$$

Therefore, the total number of edges in the flat HRM is $\bar{E}^{(h_r)} = 2^{h_r}$. In contrast, the total number of edges in the HRM grows linearly: $\sum_{i=1}^{h_r} N^{(i)}(U^{(i)} - 1)E^{(i)} = \sum_{i=1}^{h_r} 1(3-1)1 = 2h_r$.

We have thus shown that there are HRMs whose equivalent flat HRM has a number of states and edges that grows exponentially with the height of the root. $\square$

Using the aforementioned intuition, we observe that the hierarchical structure is actually expected to be useful: the number of calls across RMs with height i is $(U^{(i)} - 1)E^{(i)} = (3-1)1 = 2$, which is greater than the number of RMs with height $i-1$ (only 1).

There are cases where having $h_r > 1$ is not beneficial. For instance, given an HRM whose root has height h_r and parameterized by $N^{(i)} = 1$, $U^{(i)} = 2$ and $E^{(i)} = 1$, the number of states in the equivalent flat HRM is constant (2), whereas in the HRM itself it grows linearly with h_r. The same occurs with the number of edges. By checking the previously introduced intuition, we observe that $(U^{(i)} - 1)E^{(i)}N^{(i)} = (2-1) \cdot 1 \cdot 1 = 1 \not> N^{(i-1)} = 1$, which verifies that having RMs with multiple heights is not useful.

D Policy Learning Implementation Details

In this appendix, we describe some implementation details that were omitted in Section 4 for simplicity. First, we start by describing some methods used in policy learning. Second, we explain the option selection algorithm step-by-step and provide examples to ease its understanding.

D.1 Policies

Deep Q-networks [DQNs, 38]. We use Double DQNs [52] for both formula and call options. The DQNs associated with formula options simply take an MDP state and output a Q-value for each action. In contrast, the DQNs associated with call options also take an RM state and a context, which are encoded as follows:

- The RM state is encoded using a one-hot vector. The size of the vector is given by the number of states in the RM.
- The context, which is either $\top$ or a DNF formula with a single disjunct/conjunction, is encoded using a vector whose size is the number of propositions $|\mathcal{P}|$. Each vector position corresponds to a proposition $p \in \mathcal{P}$ whose value depends on how p appears in the context: (i) +1 if p appears positively, (ii) -1 if p appears negatively, or (iii) 0 if p does not appear. Note that if the context is $\top$, the vector solely consists of zeros.

These DQNs output a value for each possible call in the RM; however, some of these values must be masked if the corresponding calls are not available from the RM state-context used as input. For instance, the DQN for $\mathcal{A}_0$ in Figure 1b outputs a value for $\langle \mathcal{A}_1, \neg$🐮$\rangle$, $\langle \mathcal{A}_2, \top \rangle$, $\langle \mathcal{A}_1, \top \rangle$, and $\langle \mathcal{A}_\top,$🪑$\rangle$. If the RM state was u_0^0 and the context was $\top$, only the values for the first two calls are relevant. Just like unavailable calls, we also mask unsatisfiable calls (i.e., calls whose context cannot be satisfied in conjunction with the accumulated context used as input).

To speed up learning, a subset of the Q-functions associated with formula options is updated after each step. Updating all the Q-functions after each step is costly and we observed that similar performance could be obtained with this strategy. To determine the subset, we keep an update counter c_ϕ for each Q-function Q_ϕ, and a global counter c (i.e., the total number of times Q-functions have been updated). The probability of updating Q_ϕ is:

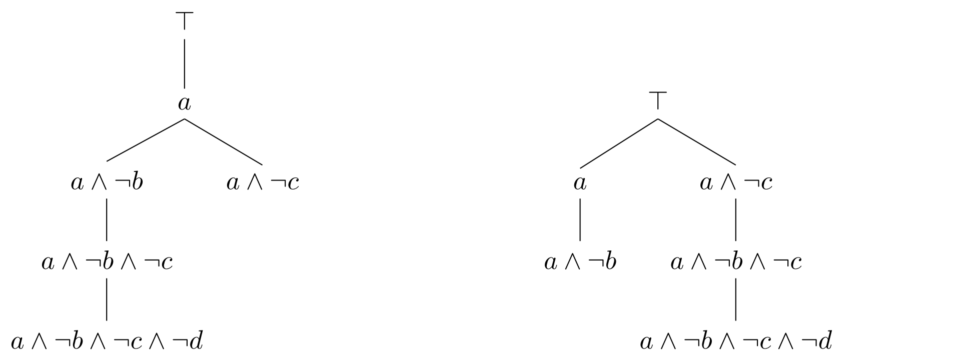
$$p_\phi = \frac{s_\phi}{\sum_{\phi'} s_{\phi'}}, \text{ where } s_\phi = c - c_\phi - 1.$$

A subset of Q-functions is chosen using this probability distribution without replacement.

Exploration. During training, the formula and call option policies are ϵ-greedy. In the case of formula options, akin to Q-functions, each option $\omega_{i,u,\Psi}^{j,\phi}$ performs exploration with an exploration factor $\epsilon_{\phi \wedge \Psi}$, which linearly decreases with the number of steps performed using the policy induced by $Q_{\phi \wedge \Psi}$. Likewise, Kulkarni et al. [28] keep an exploration factor for each subgoal, but vary it depending on the option's success rather than on the number of performed steps. In the case of call options, each RM state-context pair is associated with its own exploration factor, which linearly decreases as options started from that pair terminate.

The Formula Tree. As explained in Section 4, each formula option's policy is induced by a Q-function associated with a formula. In domains where certain proposition sets cannot occur, it is unnecessary to consider formulas that cover some of these sets. For instance, in a domain where two propositions a and b cannot be simultaneously observed (i.e., it is impossible to observe $\{a, b\}$), formulas such as $a \wedge \neg b$ or $b \wedge \neg a$ could instead be represented by the more abstract formulas a or b; therefore, $a \wedge \neg b$ and a could be both associated with a Q-function Q_a, whereas $b \wedge \neg a$ and b could be both associated with a Q-function Q_b. By reducing the number of Q-functions, the learning naturally becomes more efficient.

We represent relationships between formulas using a *formula tree* which, as the name suggests, arranges a set of formulas in a tree structure. Formally, given a set of propositions $\mathcal{P}$, a formula tree is a tuple $\mathcal{T} = \langle \mathcal{F}, \mathcal{F}_r, M \rangle$, where $\mathcal{F}$ is a set of nodes, each associated with a formula; $\mathcal{F}_r \in \mathcal{F}$ is the root of the tree and it is associated with the formula $\top$; and $M \subseteq (2^{\mathcal{P}})^*$ is a set of labels. All the nodes in the tree except for the root are associated with conjunctions. Let $X_L \subseteq 2^{2^{\mathcal{P}}}$ denote the set of literals of a formula X (e.g., if $X = a \wedge \neg b$, then $X_L = \{a, \neg b\}$). A formula X *subsumes*


$\top$
a
$a \wedge \neg b$
$a \wedge \neg c$
$a \wedge \neg b \wedge \neg c$
$a \wedge \neg b \wedge \neg c \wedge \neg d$
$\top$
a
$a \wedge \neg c$
$a \wedge \neg b$
$a \wedge \neg b \wedge \neg c$
$a \wedge \neg b \wedge \neg c \wedge \neg d$

(a) Formula tree for $M = \{\{a\}, \{b\}, \{c\}, \{d\}\}$. (b) Formula tree for $M = \{\{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}\}$.

Figure 7: Examples of formula trees. Note that the node $a \wedge \neg b \wedge \neg c$ in (a) could also be a child of $a \wedge \neg c$ (the parent depends on the insertion order).

a formula Y if (1) $X = \top$, or (2.i) $X_L \subseteq Y_L$ and (2.ii) for all labels $L \in M$, either $L \models X$ and $L \models Y$, or $L \not\models X$ and $L \not\models Y$. Case (2) indicates that Y is a special case of X (it adds more literals but it is satisfied by exactly the same labels). The tree is organized such that the formula at a given node subsumes all its descendants. The set of Q-functions is determined by the children of the root.

During the agent-environment interaction, the formula tree is updated if (i) a new formula appears in the learned HRMs, or (ii) a new label is observed. Algorithm 1 contains the pseudo-code for updating the tree in these two cases. When a new formula is added (line 1), we create a node for the formula (line 2) and add it to the tree. The insertion place is determined by exploring the tree top-down from the root $\mathcal{F}_r$ (lines 3-19). First, we check whether a child of the current node subsumes the new node (line 7). If such a node exists, then we go down this path (lines 8-9); otherwise, the new node is going to be a child of the current node (lines 16-17). In the latter case, in addition, all those children nodes from the current node that are subsumed by the new node need to become children of the new node (lines 11-15). The other core case in which the tree may need an update occurs when a new label is observed (lines 20-25) since we need to make sure that parenting relationships comply with the set of labels M. First, we find nodes inconsistent with the new label: a parenting relationship is broken (line 39) when the formula of the parent non-root node is satisfied by the label but the formula of the child node is not (or vice versa). Once the inconsistent nodes are found, we remove their current parenting relationship (lines 45-46) and reinsert them in the tree (line 47). Figure 7 shows two simple examples of formula trees, where the Q-functions are Q_a in (a), and Q_a and $Q_{a \wedge \neg c}$ in (b).

D.2 Option Selection Algorithm

Algorithm 2 shows how options are selected, updated and interrupted during an episode. Lines 1-3 correspond to the algorithm's initialization. The initial state is that of the environment, while the initial hierarchy state is formed by the root RM $\mathcal{A}_r$, its initial state u_r^0, an empty context (i.e., $\Psi = \top$), and an empty call stack. The option stack $\Omega_\mathcal{H}$ contains the options we are currently running, where options at the front are the shallowest ones (e.g., the first option in the list is taken in the root RM). The steps taken during an episode are shown in lines 4-14, which are grouped as follows:

1. The agent fills the option stack $\Omega_\mathcal{H}$ by selecting options in the HRM from the current hierarchy state until a formula option is chosen (lines 15-25). The context is propagated and augmented through the HRM (i.e., the context of the calls is conjuncted with the propagating context and converted into DNF form). Note that the context is initially $\top$ (true), and not that of the hierarchy state. It is possible that no new options are selected if the formula option chosen in a previous step has not terminated yet.
2. The agent chooses an action according to the last option in the option stack (line 6), which will always be a formula option whose policy maps states into actions. The action is applied, and the agent gets the next state (line 7). The next hierarchy state is obtained by applying the hierarchical transition function $\delta_\mathcal{H}$ using label $\mathcal{L}(\sigma_{t+1}^\Sigma)$ (line 8). The Q-functions associated with formula options' policies are updated after this step (line 9).

Algorithm 1 Formula tree operations

```
Input: A formula tree $\mathcal{T} = \langle \mathcal{F}, \mathcal{F}_r, M \rangle$, where $\mathcal{F}$ is a set of nodes, $\mathcal{F}_r \in \mathcal{F}$ is the root node
    (associated with the formula $\top$), and $M$ is a set of labels.
 1: procedure AddFormula($f$)
 2:     AddNode(CreateNode($f$))
 3: procedure AddNode(new_node)
 4:     current_node $\leftarrow \mathcal{F}_r$
 5:     added_node $\leftarrow \bot$
 6:     while added_node $= \bot$ do
 7:         child_node $\leftarrow$ FindSubsumingChild(current_node, new_node)
 8:         if child_node $\neq$ nil then                        ▷ Keep exploring down this path
 9:             current_node $\leftarrow$ child_node
10:         else                                                         ▷ Insert the node
11:             subsumed_children $\leftarrow$ GetSubsumedChildren(current_node, new_node)
12:             new_node.children $\leftarrow$ new_node.children $\cup$ subsumed_children
13:             for child $\in$ subsumed_children do
14:                 current_node.children $\leftarrow$ current_node.children $\setminus$ {child}
15:                 child.parent $\leftarrow$ new_node
16:             current_node.children $\leftarrow$ current_node.children $\cup$ {new_node}
17:             new_node.parent $\leftarrow$ current_node
18:             added_node $\leftarrow \top$
19:     $\mathcal{F} \leftarrow \mathcal{F} \cup$ {new_node}
20: procedure OnLabel($L$)
21:     $M \leftarrow M \cup \{L\}$
22:     inconsistent_nodes $\leftarrow$ {}
23:     for child $\in \mathcal{F}_r$.children do
24:         FindInconsistentNodes(child, $L$, inconsistent_nodes)
25:     ReinsertInconsistentNodes(inconsistent_nodes)
26: procedure FindSubsumingChild(current_node, new_node)
27:     for child $\in$ current_node.children do
28:         if child.formula subsumes new_node.formula then
29:             return child
30:     return nil
31: procedure GetSubsumedChildren(current_node, new_node)
32:     subsumed_children $\leftarrow$ {}
33:     for child $\in$ current_node.children do
34:         if new_node.formula subsumes child.formula then
35:             subsumed_children $\leftarrow$ subsumed_children $\cup$ {new_node}
36:     return subsumed_children
37: procedure FindInconsistentNodes(current_node, $L$, inconsistent_nodes)
38:     for child $\in$ current_node.children do
39:         if $L \models$ current_node.formula $\oplus L \models$ child.formula then
40:             inconsistent_nodes $\leftarrow$ inconsistent_nodes $\cup$ {child}
41:         else
42:             FindInconsistentNodes(child, $L$, inconsistent_nodes)
43: procedure ReinsertInconsistentNodes(inconsistent_nodes)
44:     for node $\in$ inconsistent_nodes do
45:         node.parent.children $\leftarrow$ node.parent.children $\setminus$ {node}
46:         node.parent $\leftarrow$ nil
47:         AddNode(node)
```

3. The option stack $\Omega_{\mathcal{H}}$ is updated by removing those options that have terminated (lines 10, 26-45). The terminated options are saved in a different list Ω_{β} to update the Q-functions of the RMs where they were initiated later on (line 11). The termination of the options is performed as described in Section 4. All options terminate if a terminal state is reached (lines 27-28). Otherwise, we check options in $\Omega_{\mathcal{H}}$ from deeper to shallower levels. The first checked option is always a formula option, which terminates if the hierarchy state has changed (line 40). In contrast, a call option terminates if it does not appear in the stack (lines 33, 46-51).[4] When an option is found to terminate, it is added to Ω_{β} and removed from $\Omega_{\mathcal{H}}$ (lines 35-36, 41-42). If a non-terminating option is found (lines 37, 43), we stop checking for termination (no higher level options can have terminated in this case).

4. If at least one option has terminated (line 12), the option stack is updated such that it contains all options appearing in the call stack (lines 13, 52-70). Options are derived for the full stack if $\Omega_{\mathcal{H}}$ is empty (lines 53, 54), or for the part of the stack not appearing in $\Omega_{\mathcal{H}}$ (lines 56-59). The new derived options (lines 61-70) from the call stack are assumed to start in the same state as the last terminated option (i.e., the shallowest terminated option, line 63) and to have been run for the same number of steps too. Crucially, the contexts should be propagated accordingly, starting from the context of the last terminated option (line 69).

   As a result of the definition of the hierarchical transition function $\delta_{\mathcal{H}}$, the contexts in the stack may be DNF formulas with more than one disjunct. In contrast, the contexts associated with options are either $\top$ or DNFs with a single disjunct (remember that an option is formed for each disjunct). For instance, this occurs if the context is $a \vee b$ and $\{a, b\}$ is observed: since both disjuncts are satisfied, the context shown in the call stack will be the full disjunction $a \vee b$. In the simplest case, the derived option (which as said before is associated with a DNF with a single disjunct or $\top$) can include one of these disjuncts chosen uniformly at random (line 67). Alternatively, we could memorize all the derived options and perform identical updates for both later on once terminated.

D.3 Examples

We briefly describe some examples of how policy learning is performed in the HRM of Figure 1b. We first enumerate the options in the hierarchy. The formula options are $\omega_{1,0,\neg 🐮}^{\top,🌿}$, $\omega_{2,0,\top}^{\top,🐮}$, $\omega_{1,0,\top}^{\top,🌿}$, $\omega_{1,1,\top}^{\top,🪑}$, $\omega_{2,1,\top}^{\top,🪑}$, and $\omega_{0,3,\top}^{\top,🎓}$. The first option should lead the agent to observe the label $\{🌿\}$ to satisfy $🌿 \wedge \neg 🐮$. The Q-functions associate with this set of options are $Q_{🌿 \wedge \neg 🐮}$, $Q_{🐮}$, $Q_{🌿}$, $Q_{🪑}$ and $Q_{🎓}$. Note that $\omega_{1,1,\top}^{\top,🪑}$ and $\omega_{2,1,\top}^{\top,🪑}$ are both associated with $Q_{🪑}$. Conversely, the call options are $\omega_{0,0,\top}^{1,\neg 🐮}$, $\omega_{0,0,\top}^{2,\top}$, $\omega_{0,1,\top}^{2,\top}$, and $\omega_{0,2,\top}^{1,\top}$, where the first one achieves its local goal if formula options $\omega_{1,0,\neg 🐮}^{\top,🌿}$ and $\omega_{1,1,\top}^{\top,🪑}$ sequentially achieve theirs. The associated Q-functions are Q_0, Q_1 and Q_2. Note that $\omega_{0,0,\top}^{2,\top}$ and $\omega_{0,1,\top}^{2,\top}$ are both associated with Q_2.

We now describe a few steps of the aforementioned option selection algorithm in two scenarios. First, we consider the scenario where all chosen options are run to completion (i.e., until their local goals are achieved):

1. The initial hierarchy state is $\langle \mathcal{A}_0, u_0^0, \top, [] \rangle$ and the option stack $\Omega_{\mathcal{H}}$ is empty. We select options to fill $\Omega_{\mathcal{H}}$. The first option is chosen from u_0^0 in $\mathcal{A}_0$ using a policy induced by Q_0. At this state, the available options are $\omega_{0,0,\top}^{1,\neg 🐮}$ and $\omega_{0,0,\top}^{2,\top}$. Let's assume that the former is chosen. Then an option from the initial state of $\mathcal{A}_1$ under context $\neg 🐮$ is chosen, which can only be $\omega_{1,0,\neg 🐮}^{\top,🌿}$. Since this option is a formula option (the call is made to $\mathcal{A}_{\top}$), we do not select any more options and the option stack is $\Omega_{\mathcal{H}} = \langle \omega_{0,0,\top}^{1,\neg 🐮}, \omega_{1,0,\neg 🐮}^{\top,🌿} \rangle$.

2. The agent selects options according to the formula option in $\Omega_{\mathcal{H}}$, $\omega_{1,0,\neg 🐮}^{\top,🌿}$, whose policy is induced by $Q_{🌿 \wedge \neg 🐮}$. Let us assume that the policy tells the agent to turn right. Since the label at this location is empty, the hierarchy state remains the same; therefore, no options terminate, and the option stack will remain the same.

[4]We denote by $\phi_1 \subseteq \phi_2$, where $\phi_1, \phi_2 \in \text{DNF}_{\mathcal{P}}$, the fact that all the disjuncts of ϕ_1 appear in ϕ_2. This containment relationship also holds if both formulas are $\top$. For instance, $(a \wedge \neg c) \subseteq (a \wedge \neg c) \vee d$.

Algorithm 2 Episode execution using an HRM (continues in p. 27)

```
Input: An HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P} \rangle$ and an environment ENV $= \langle \Sigma, A, p, r, \gamma, \mathcal{P}, \mathcal{L}, \tau \rangle$.
 1: $\boldsymbol{\sigma}_0 \leftarrow$ ENV.INIT()                                         ▷ Initial MDP tuple
 2: $\langle \mathcal{A}_i, u, \Psi, \Gamma \rangle \leftarrow \langle \mathcal{A}_r, u_r^0, \top, [] \rangle$                              ▷ Initial hierarchy state
 3: $\Omega_\mathcal{H} \leftarrow []$                                                   ▷ Initial option stack
 4: for each step $t = 0, \ldots,$ do
 5:     $\Omega_\mathcal{H} \leftarrow$ FILLOPTIONSTACK($\sigma_t^\Sigma, \langle \mathcal{A}_i, u, \Psi, \Gamma \rangle, \Omega_\mathcal{H}$)          ▷ Expand the option stack
 6:     $a \leftarrow$ SELECTACTION($\sigma_t^\Sigma, \Omega_\mathcal{H}$)        ▷ Choose $a$ according to the last option in $\Omega_\mathcal{H}$
 7:     $\boldsymbol{\sigma}_{t+1} \leftarrow$ ENV.APPLYACTION($a$)
 8:     $\langle \mathcal{A}_j, u', \Psi', \Gamma' \rangle \leftarrow \delta_\mathcal{H}(\langle \mathcal{A}_i, u, \Psi, \Gamma \rangle, \mathcal{L}(\sigma_{t+1}^\Sigma))$             ▷ Apply transition function
 9:     UPDATEFORMULAQFUNCTIONS($\boldsymbol{\sigma}_t, a, \boldsymbol{\sigma}_{t+1}$)
10:     $\Omega_\beta, \Omega_\mathcal{H} \leftarrow$ TERMINATEOPTIONS($\Omega_\mathcal{H}, \boldsymbol{\sigma}, \langle \mathcal{A}_i, u, \Psi, \Gamma \rangle, \langle \mathcal{A}_j, u', \Psi', \Gamma' \rangle$)
11:     UPDATECALLQFUNCTIONS($\Omega_\beta, \sigma_{t+1}^\Sigma, \mathcal{L}(\sigma_{t+1}^\Sigma)$)
12:     if $|\Omega_\beta| > 0$ then
13:         $\Omega_\mathcal{H} \leftarrow$ ALIGNOPTIONSTACK($\Omega_\mathcal{H}, \Gamma', \Omega_\beta$)
14:     $\langle \mathcal{A}_i, u, \Psi, \Gamma \rangle \leftarrow \langle \mathcal{A}_j, u', \Psi', \Gamma' \rangle$
15: procedure FILLOPTIONSTACK($\sigma^\Sigma, \langle \mathcal{A}_i, u, \cdot, \Gamma \rangle, \Omega_\mathcal{H}$)
16:     $\Omega'_\mathcal{H} \leftarrow \Omega_\mathcal{H}$
17:     $\Psi \leftarrow \top$                                                ▷ The context is initially true
18:     $\mathcal{A}_j \leftarrow \mathcal{A}_i; v \leftarrow u$                 ▷ The state-automaton pair in which an option is selected
19:     while the last option in $\Omega'_\mathcal{H}$ is not a formula option do
20:         $\omega_{j,v,\Psi}^{x,\phi} \leftarrow$ SELECTOPTION($\sigma^\Sigma, \mathcal{A}_j, v, \Psi$)       ▷ Select an option (e.g., with $\epsilon$-greedy)
21:         if $x \neq \top$ then                                         ▷ If the option is a call option
22:             $\mathcal{A}_j \leftarrow \mathcal{A}_x; v \leftarrow u_x^0$           ▷ Next option is chosen on the called RM's initial state
23:             $\Psi \leftarrow$ DNF($\Psi \wedge \phi$)                                     ▷ Update the context
24:         $\Omega'_\mathcal{H} \leftarrow \Omega_\mathcal{H} \oplus \omega_{j,v,\Psi}^{x,\phi}$                ▷ Update the option stack (concatenate new option)
25:     return $\Omega'_\mathcal{H}$
26: procedure TERMINATEOPTIONS($\Omega_\mathcal{H}, \boldsymbol{\sigma}, \langle \mathcal{A}_i, u, \Psi, \Gamma \rangle, \langle \mathcal{A}_j, u', \Psi', \Gamma' \rangle$)
27:     if $\sigma^T = \top$ then
28:         return $\Omega_\mathcal{H}, []$                                        ▷ All options terminate
29:     $\Omega_\beta \leftarrow []; \Omega'_\mathcal{H} \leftarrow \Omega_\mathcal{H}$                                  ▷ Initialize structures
30:     while $|\Omega'_\mathcal{H}| > 0$ do                              ▷ While the option stack is not empty
31:         $\omega_{k,v,\Phi}^{x,\phi} \leftarrow$ last option in $\Omega'_\mathcal{H}$
32:         if $x \neq \top$ then                                         ▷ If the option is a call option
33:             in_stack, _ $\leftarrow$ OPTIONINSTACK($\omega_{k,v,\Phi}^{x,\phi}, \Gamma'$)
34:             if $\neg$in_stack then
35:                 $\Omega_\beta \leftarrow \Omega_\beta \oplus \omega_{k,v,\Phi}^{x,\phi}$                  ▷ Update the list of terminated options
36:                 $\Omega'_\mathcal{H} \leftarrow \Omega'_\mathcal{H} \ominus \omega_{k,v,\Phi}^{x,\phi}$             ▷ Remove the last option from the option stack
37:             else
38:                 break                                                  ▷ Stop unrolling
39:         else
40:             if $\langle \mathcal{A}_i, u, \Psi, \Gamma \rangle \neq \langle \mathcal{A}_j, u', \Psi', \Gamma' \rangle$ then         ▷ If the hierarchy state has changed...
41:                 $\Omega_\beta \leftarrow \Omega_\beta \oplus \omega_{k,v,\Phi}^{x,\phi}$                  ▷ Update the list of terminated options
42:                 $\Omega'_\mathcal{H} \leftarrow \Omega'_\mathcal{H} \ominus \omega_{k,v,\Phi}^{x,\phi}$             ▷ Remove the last option from the option stack
43:             else
44:                 break                                                  ▷ Stop unrolling
45:     return $\Omega_\beta, \Omega'_\mathcal{H}$
46: procedure OPTIONINSTACK($\omega_{k,v,\Psi}^{x,\phi}, \Gamma$)
47:     for $l = 0 \ldots |\Gamma| - 1$ do
48:         $\langle u_f, \cdot, \mathcal{A}_i, \mathcal{A}_j, \phi', \Psi' \rangle \leftarrow \Gamma_l$
49:         if $u_f = v \wedge i = k \wedge j = x \wedge \phi \subseteq \phi' \wedge \Psi \subseteq \Psi'$ then ▷ The call option is in the call stack
50:             return $\top, l$                     ▷ Return whether it appears in the stack and the index
51:     return $\bot, -1$
```

```
52: procedure AlignOptionStack($\Omega_{\mathcal{H}}, \Gamma, \Omega_\beta$)
53:     if $|\Omega_{\mathcal{H}}| = 0$ then
54:         return AlignOptionStackHelper($\Omega_{\mathcal{H}}, \Gamma, \Omega_\beta, 0$)
55:     else
56:         $\omega_{k,v,\Psi}^{x,\phi} \leftarrow$ last option in $\Omega_{\mathcal{H}}$
57:         in_stack, stack_index $\leftarrow$ OptionInStack($\omega_{k,v,\Psi}^{x,\phi}, \Gamma$)
58:         if in_stack then
59:             return AlignOptionStackHelper($\Omega_{\mathcal{H}}, \Gamma, \Omega_\beta$, stack_index)
60:     return $\Omega_{\mathcal{H}}$
61: procedure AlignOptionStackHelper($\Omega_{\mathcal{H}}, \Gamma, \Omega_\beta$, stack_index)
62:     $\Omega'_{\mathcal{H}} \leftarrow \Omega_{\mathcal{H}}$
63:     $\omega_{\cdot,\cdot,\Psi}^{\cdot,\cdot} \leftarrow$ last option in $\Omega_\beta$          ▷ Shallowest terminated option
64:     $\Psi' \leftarrow \Psi$                                        ▷ Context initialized from last option
65:     for $l =$ stack_index $\ldots |\Gamma| - 1$ do
66:         $\langle u_f, \cdot, \mathcal{A}_i, \mathcal{A}_j, \phi, \cdot \rangle \leftarrow \Gamma_l$
67:         $\phi_{sel} \leftarrow$ Select disjunct from $\phi$ (e.g., randomly)
68:         $\Omega'_{\mathcal{H}} \leftarrow \Omega'_{\mathcal{H}} \oplus \omega_{i,u_f,\Psi'}^{j,\phi_{sel}}$          ▷ Append new option to the option stack
69:         $\Psi' \leftarrow$ DNF($\Psi' \wedge \phi$)
70:     return $\Omega'_{\mathcal{H}}$
```

3. Let us assume that the agent moves forward twice, thus observing $\{\text{🎋}\}$. The hierarchy state then becomes $\langle \mathcal{A}_1, u_1^1, \top, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top\rangle]\rangle$ (see Appendix B for a step-by-step application of the hierarchical transition function). We check which options in $\Omega_{\mathcal{H}}$ have terminated starting from the last chosen one. The formula option $\omega_{1,0,\neg\text{🐮}}^{\top,\text{🎋}}$ terminates because the hierarchy state has changed. In contrast, the call option $\omega_{0,0,\top}^{1,\neg\text{🐮}}$ does not terminate since there is an item in the call stack, $\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top\rangle$ that can be mapped into it (meaning that the option is running).
4. An experience $\langle \boldsymbol{\sigma}, \omega_{1,0,\neg\text{🐮}}^{\top,\text{🎋}}, \boldsymbol{\sigma}'\rangle$ is formed for the terminated option, where $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ are the observed tuples on initiation and termination respectively. This tuple is added to the replay buffer associated with the RM where the option appears, $\mathcal{D}_1$, since it achieved its goal (i.e., a label that satisfied $\text{🎋} \wedge \neg\text{🐮}$ was observed).
5. We align $\Omega_{\mathcal{H}}$ with the new stack. In this case, $\Omega_{\mathcal{H}}$ remains unchanged since its only option can be mapped into an item of the new stack.
6. We start a new step. Since the option stack does not contain a formula option, we select new options from the current hierarchy state according to a policy induced by Q_1. In this case, there is a single eligible option: $\omega_{1,1,\top}^{\top,\text{🪑}}$.

In the second scenario, we observe what occurs when the HRM traversal differs from the options chosen by the agent:

1. The initial step is like the one in the previous scenario, but we assume $\omega_{0,0,\top}^{2,\top}$ is selected instead. Then, since this is a call option, an option from the initial state of $\mathcal{A}_2$ under context $\top$ is chosen, which can only be $\omega_{2,0,\top}^{\top,\text{🐮}}$. The option stack thus becomes $\Omega_{\mathcal{H}} = \langle \omega_{0,0,\top}^{2,\top}, \omega_{2,0,\top}^{\top,\text{🐮}}\rangle$.
2. Let us assume that by taking actions according to $\omega_{2,0,\top}^{\top,\text{🐮}}$ we end up observing $\{\text{🎋}\}$. Like in the previous scenario, the hierarchy state becomes $\langle \mathcal{A}_1, u_1^1, \top, [\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top\rangle]\rangle$. We check which options in $\Omega_{\mathcal{H}}$ have terminated. The formula option $\omega_{2,0,\top}^{\top,\text{🐮}}$ terminates since the hierarchy state has changed, and the call option $\omega_{0,0,\top}^{2,\top}$ also terminates since it cannot be mapped into an item of the call stack. Note that these options should intuitively finish since the HRM is being traversed through a path different from that chosen by the agent.
3. The replay buffers are not updated for these options since they have not achieved their local goals.

4. We align $\Omega_{\mathcal{H}}$ with the new stack. The only item of the stack $\langle u_0^0, u_0^1, \mathcal{A}_0, \mathcal{A}_1, \neg\text{🐮}, \top\rangle$ can be mapped into option $\omega_{0,0,\top}^{1,\neg\text{🐮}}$. We assume that this option starts on the same tuple $\boldsymbol{\sigma}$ and that it has run for the same number of steps as the last terminated option $\omega_{0,0,\top}^{2,\top}$.

E Curriculum Learning Implementation

We here describe the details of the curriculum learning method described in Section 5. When an episode is completed for $\mathcal{M}_{ij}$, R_{ij} is updated using the episode's undiscounted return r as $R_{ij} \leftarrow \beta R_{ij} + (1-\beta)r$, where β is a hyperparameter. A score $c_{ij} = 1 - R_{ij}$ is computed from the return and used to determine the probability of selecting tasks and instances. Note that this scoring function, also used in the curriculum method by Andreas et al. [2], assumes that the undiscounted return ranges between 0 and 1 (see Section 2). The probability of choosing task i is $\max_j c_{ij} / \sum_k \max_l c_{kl}$; that is, the task for which an instance is performing very poorly has a higher probability. Having selected task i, the probability of choosing instance j is $c_{ij}/\sum_k c_{ik}$, i.e. worst-performing instances have a higher probability.

F Learning a Hierarchy of Reward Machines from Traces with ILASP

We formalize the task of learning an HRM using ILASP [30], an inductive logic programming system that learns answer set programs (ASP) from examples. We address the reader to Gelfond and Kahl [19] for an introduction to ASP, and to Law [29] for ILASP. Our formalization is close to that by Furelos-Blanco et al. [17] for flat finite-state machines. Without loss of generality, as stated in Section 5, we assume that each RM has exactly one accepting and one rejecting state. We first describe how HRMs are represented in ASP, and then explain the encoding of the HRM learning task in ILASP.

F.1 Representation of a Hierarchy of Reward Machines in Answer Set Programming

In this section, we explain how HRMs are represented using Answer Set Programming (ASP). First, we describe how traces are represented. Then, we present how HRMs themselves are represented and also introduce the general rules that describe the behavior of these hierarchies. Finally, we prove the correctness of the representation.

Definition 9 (ASP representation of a label trace)**.** Given a label trace $\lambda = \langle L_0, \ldots, L_n\rangle$, $M(\lambda)$ denotes the set of ASP facts that describe it:

$$M(\lambda) = \begin{array}{l} \{\texttt{label}(p,t).\mid 0 \leq t \leq n, p \in L_t\} \cup \\ \{\texttt{step}(t).\mid 0 \leq t \leq n\} \cup \\ \{\texttt{last}(n).\}. \end{array}$$

The $\texttt{label}(p,t)$ fact indicates that proposition $p \in \mathcal{P}$ is observed in step t, $\texttt{step}(t)$ states that t is a step of the trace, and $\texttt{last}(n)$ indicates that the trace ends in step n.

Example 2. The set of ASP facts for the label trace $\lambda = \langle\{\text{🔖}\}, \{\}, \{\text{🎓}\}\rangle$ is $M(\lambda) = \{\texttt{label}(\text{🔖}, 0).,$ $\texttt{label}(\text{🎓}, 2).,$ $\texttt{step}(0),$ $\texttt{step}(1).,$ $\texttt{step}(2).,$ $\texttt{last}(2).\}$.

Definition 10 (ASP representation of an HRM)**.** Given an HRM $\mathcal{H} = \langle\mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$, $M(\mathcal{H}) = \bigcup_{\mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}} M(\mathcal{A}_i)$, where:

$$M(\mathcal{A}_i) = M_U(\mathcal{A}_i) \cup M_\varphi(\mathcal{A}_i),$$
$$M_U(\mathcal{A}_i) = \{\texttt{state}(u, \mathcal{A}_i).\mid u \in U_i\},$$

$$M_\varphi(\mathcal{A}_i) = \left\{ \begin{array}{l|c} \texttt{call}(u, u', x+e, \mathcal{A}_i, \mathcal{A}_j). & \\ \bar{\varphi}(u, u', x+e, \mathcal{A}_i, \texttt{T})\,\texttt{:-}\,\texttt{not label}(p_1, \texttt{T}), \texttt{step}(\texttt{T}). & \mathcal{A}_j \in \mathcal{A}, u, u' \in U_i, \\ \vdots & \varphi_i(u, u', \mathcal{A}_j) \neq \bot, \\ \bar{\varphi}(u, u', x+e, \mathcal{A}_i, \texttt{T})\,\texttt{:-}\,\texttt{not label}(p_n, \texttt{T}), \texttt{step}(\texttt{T}). & x = \sum_{k=0}^{j-1} |\varphi_i(u, u', \mathcal{A}_k)|, \\ \bar{\varphi}(u, u', x+e, \mathcal{A}_i, \texttt{T})\,\texttt{:-}\,\texttt{label}(p_{n+1}, \texttt{T}), \texttt{step}(\texttt{T}). & e \in [1, |\varphi_i(u, u', \mathcal{A}_j)|], \\ \vdots & \phi_e \in \varphi_i(u, u', \mathcal{A}_j), \\ \bar{\varphi}(u, u', x+e, \mathcal{A}_i, \texttt{T})\,\texttt{:-}\,\texttt{label}(p_m, \texttt{T}), \texttt{step}(\texttt{T}). & \phi_e = p_1 \wedge \cdots \wedge p_n \\ & \wedge \neg p_{n+1} \wedge \cdots \wedge \neg p_m \end{array} \right\}.$$

Note that each non-leaf RM $\mathcal{A}_i$ in the hierarchy is associated with its own set of rules $M(\mathcal{A}_i)$, which are described as follows:

- Facts $\texttt{state}(u, \mathcal{A}_i)$ indicate that u is a state of RM $\mathcal{A}_i$.
- Facts $\texttt{call}(u, u', e, \mathcal{A}_i, \mathcal{A}_j)$ indicate that edge e between states u and u' in RM $\mathcal{A}_i$ is labeled with a call to RM $\mathcal{A}_j$.
- Normal rules whose *head* is of the form $\bar{\varphi}(u, u', e, \mathcal{A}_i, \texttt{T})$ indicate that the transition from state u to u' with edge e in RM $\mathcal{A}_i$ does not hold at step $\texttt{T}$. The *body* of these rules consists of a single $\texttt{label}(p, \texttt{T})$ literal and a $\texttt{step}(\texttt{T})$ atom indicating that $\texttt{T}$ is a step. Commonly, variables are represented using upper case letters in ASP, which is the case of steps $\texttt{T}$ here.

There are some important things to take into account regarding the encoding:

- There is no leaf RM $\mathcal{A}_\top$. We later introduce the ASP rules to emulate it.
- The edge identifiers e between a given pair of states (u, u') range from 1 to the total number of conjunctions/disjuncts between them. Note that in $M_\varphi(\mathcal{H})$ we assume that the leaf RM has an index, just like the other RMs in the HRM. The index could be n since the rest are numbered from 0 to $n-1$.

Example 3. The following rules represent the HRM in Figure 1b:

$$\left\{\begin{array}{ll}\texttt{state}(u_0^0,\mathcal{A}_0).\ \texttt{state}(u_0^1,\mathcal{A}_0).\ \texttt{state}(u_0^2,\mathcal{A}_0). & \texttt{state}(u_0^3,\mathcal{A}_0).\ \texttt{state}(u_0^A,\mathcal{A}_0). \\ \texttt{call}(u_0^0,u_0^1,1,\mathcal{A}_0,\mathcal{A}_1).\ \texttt{call}(u_0^0,u_0^2,1,\mathcal{A}_0,\mathcal{A}_2). & \texttt{call}(u_0^1,u_0^3,1,\mathcal{A}_0,\mathcal{A}_2).\ \texttt{call}(u_0^2,u_0^3,1,\mathcal{A}_0,\mathcal{A}_1). \\ \texttt{call}(u_0^3,u_0^A,1,\mathcal{A}_0,\mathcal{A}_\top). & \bar{\varphi}(u_0^0,u_0^1,1,\mathcal{A}_0,\texttt{T})\texttt{:-}\,\texttt{label}(\text{🐮},\texttt{T}),\texttt{step}(\texttt{T}). \\ \bar{\varphi}(u_0^3,u_0^A,1,\mathcal{A}_0,\texttt{T})\texttt{:-}\,\texttt{not label}(\text{🛖},\texttt{T}),\texttt{step}(\texttt{T}). & \end{array}\right\}\cup$$
$$\left\{\begin{array}{ll}\texttt{state}(u_1^0,\mathcal{A}_1).\ \texttt{state}(u_1^1,\mathcal{A}_1).\ \texttt{state}(u_1^A,\mathcal{A}_1). & \texttt{call}(u_1^0,u_1^1,1,\mathcal{A}_1,\mathcal{A}_\top).\ \texttt{call}(u_1^1,u_1^A,1,\mathcal{A}_1,\mathcal{A}_\top). \\ \bar{\varphi}(u_1^0,u_1^1,1,\mathcal{A}_1,\texttt{T})\texttt{:-}\,\texttt{not label}(\text{🌱},\texttt{T}),\texttt{step}(\texttt{T}). & \bar{\varphi}(u_1^1,u_1^A,1,\mathcal{A}_1,\texttt{T})\texttt{:-}\,\texttt{not label}(\text{🏠},\texttt{T}),\texttt{step}(\texttt{T}). \end{array}\right\}\cup$$
$$\left\{\begin{array}{ll}\texttt{state}(u_2^0,\mathcal{A}_2).\ \texttt{state}(u_2^1,\mathcal{A}_2).\ \texttt{state}(u_2^A,\mathcal{A}_2). & \texttt{call}(u_2^0,u_2^1,1,\mathcal{A}_2,\mathcal{A}_\top).\ \texttt{call}(u_2^1,u_2^A,1,\mathcal{A}_2,\mathcal{A}_\top). \\ \bar{\varphi}(u_2^0,u_2^1,1,\mathcal{A}_2,\texttt{T})\texttt{:-}\,\texttt{not label}(\text{🐮},\texttt{T}),\texttt{step}(\texttt{T}). & \bar{\varphi}(u_2^1,u_2^A,1,\mathcal{A}_2,\texttt{T})\texttt{:-}\,\texttt{not label}(\text{🏠},\texttt{T}),\texttt{step}(\texttt{T}). \end{array}\right\}.$$

General Rules. The following sets of rules, whose union is denoted by $R = \cup_{i=0}^{5} R_i$, represent how an HRM functions (e.g., how transitions are taken or the acceptance/rejection criteria). For simplicity, all initial, accepting and rejecting states are denoted by u^0, u^A and u^R respectively.

The rule below is the inversion of the negation of the state transition function $\bar{\varphi}$. Note that the predicate for φ includes the called RM $\texttt{M2}$ as an argument.

$$R_0 = \{\varphi(\texttt{X},\texttt{Y},\texttt{E},\texttt{M},\texttt{M2},\texttt{T})\texttt{:-}\,\texttt{not}\ \bar{\varphi}(\texttt{X},\texttt{Y},\texttt{E},\texttt{M},\texttt{T}),\texttt{call}(\texttt{X},\texttt{Y},\texttt{E},\texttt{M},\texttt{M2}),\texttt{step}(\texttt{T}).\}\,.$$

The rule set R_1 introduces the $\texttt{pre\_sat}(\texttt{X},\texttt{M},\texttt{T})$ predicate, which encodes the exit condition presented in Section 3 and indicates whether a call from state $\texttt{X}$ of RM $\texttt{M}$ can be started at time $\texttt{T}$. The first rule corresponds to the base case and indicates that if the leaf $\mathcal{A}_\top$ is called then the condition is satisfied if the associated formula is satisfied. The second rule applies to calls to non-leaf RMs, where we need to satisfy the context of the call (like in the base case), and also check whether a call from the initial state of the potentially called RM can be started.

$$R_1 = \left\{\begin{array}{l}\texttt{pre\_sat}(\texttt{X},\texttt{M},\texttt{T})\texttt{:-}\,\varphi(\texttt{X},\_,\_,\texttt{M},\mathcal{A}_\top,\texttt{T}). \\ \texttt{pre\_sat}(\texttt{X},\texttt{M},\texttt{T})\texttt{:-}\,\varphi(\texttt{X},\_,\_,\texttt{M},\texttt{M2},\texttt{T}),\texttt{pre\_sat}(u^0,\texttt{M2},\texttt{T}),\texttt{M2!=}\mathcal{A}_\top.\end{array}\right\}.$$

The rule set R_2 introduces the `reachable(X,M,T0,T2)` predicate, which indicates that state `X` of RM `M` is reached between steps `T0` and `T2`. The latter step can also be seen as the step we are currently at. The first fact indicates that the initial state of the root RM is reached from step `0` to step `0`. The second rule indicates that the initial state of a non-root RM is reached from step `T` to step `T` (i.e., it is reached anytime). The third rule represents the loop transition in the initial state of the root $\mathcal{A}_r$: we stay there if no call can be started at `T` (i.e., we are not moving in the HRM). The fourth rule is analogous to the third but for the accepting state of the root instead of the initial state. Remember this is the only accepting state in the HRM that does not return control to the calling RM. The fifth rule is also similar to the previous ones: it applies to states reached after `T0` that are non-accepting, which excludes looping in initial states of non-root RMs at the time of starting them (i.e., loops are permitted in the initial state of a non-root RM if we can reach it afterwards by going back to it). The last rule indicates that `Y` is reached at step `T2` in RM `M` started at `T0` if there is an outgoing transition from the current state X to Y at time T that holds between T and T2, and state X has been reached between T0 and T. We will later see how δ is defined.

$$R_2 = \left\{\begin{array}{l} \texttt{reachable}(u^0, \mathcal{A}_r, \texttt{0}, \texttt{0}). \\ \texttt{reachable}(u^0, \texttt{M}, \texttt{T}, \texttt{T})\texttt{:-}\,\texttt{state}(u^0, \texttt{M}), \texttt{M!=}\mathcal{A}_r, \texttt{step}(\texttt{T}). \\ \texttt{reachable}(\texttt{X}, \texttt{M}, \texttt{T0}, \texttt{T+1})\texttt{:-}\,\texttt{reachable}(\texttt{X}, \texttt{M}, \texttt{T0}, \texttt{T}), \texttt{not pre\_sat}(\texttt{X}, \texttt{M}, \texttt{T}), \\ \qquad\qquad\qquad\qquad\qquad\quad \texttt{step}(\texttt{T}), \texttt{X=}u^0, \texttt{M=}\mathcal{A}_r. \\ \texttt{reachable}(\texttt{X}, \texttt{M}, \texttt{T0}, \texttt{T+1})\texttt{:-}\,\texttt{reachable}(\texttt{X}, \texttt{M}, \texttt{T0}, \texttt{T}), \texttt{not pre\_sat}(\texttt{X}, \texttt{M}, \texttt{T}), \\ \qquad\qquad\qquad\qquad\qquad\quad \texttt{step}(\texttt{T}), \texttt{X=}u^A, \texttt{M=}\mathcal{A}_r. \\ \texttt{reachable}(\texttt{X}, \texttt{M}, \texttt{T0}, \texttt{T+1})\texttt{:-}\,\texttt{reachable}(\texttt{X}, \texttt{M}, \texttt{T0}, \texttt{T}), \texttt{not pre\_sat}(\texttt{X}, \texttt{M}, \texttt{T}), \\ \qquad\qquad\qquad\qquad\qquad\quad \texttt{step}(\texttt{T}), \texttt{T0<T}, \texttt{X!=}u^A. \\ \texttt{reachable}(\texttt{Y}, \texttt{M}, \texttt{T0}, \texttt{T2})\texttt{:-}\,\texttt{reachable}(\texttt{X}, \texttt{M}, \texttt{T0}, \texttt{T}), \delta(\texttt{X}, \texttt{Y}, \texttt{M}, \texttt{T}, \texttt{T2}). \end{array}\right\}.$$

The rule set R_3 introduces two predicates. The predicate satisfied(M, T0, TE) indicates that RM M is satisfied if its accepting state u^A is reached between steps T0 and TE. Likewise, the predicate failed(M, T0, TE) indicates that RM M fails if its rejecting state u^R is reached between steps T0 and TE. These two descriptions correspond to the first and third rules. The second rule applies to the leaf RM $\mathcal{A}_\top$, which always returns control immediately; thus, it is always satisfied between any two consecutive steps.

$$R_3 = \left\{\begin{array}{l} \texttt{satisfied}(\texttt{M}, \texttt{T0}, \texttt{TE})\texttt{:-}\,\texttt{reachable}(u^A, \texttt{M}, \texttt{T0}, \texttt{TE}). \\ \texttt{satisfied}(\mathcal{A}_\top, \texttt{T}, \texttt{T+1})\texttt{:-}\,\texttt{step}(\texttt{T}). \\ \texttt{failed}(\texttt{M}, \texttt{T0}, \texttt{TE})\texttt{:-}\,\texttt{reachable}(u^R, \texttt{M}, \texttt{T0}, \texttt{TE}). \end{array}\right\}$$

The following set, R_4, encodes multi-step transitions within an RM. The predicate $\delta(\texttt{X}, \texttt{Y}, \texttt{M}, \texttt{T}, \texttt{T2})$ expresses that the transition from state X to state Y in RM M is satisfied between steps T and T2. The first rule indicates that this occurs if the context labeling a call to an RM M2 is satisfied and that RM is also satisfied (i.e., its accepting state is reached) between these two steps. In contrast, the second rule is used for the case in which the rejecting state of the called RM is reached between those steps. In the latter case, we transition to the local rejecting state u^R of M (i.e., the state we would have transitioned to does not matter). This follows from the assumption that rejecting states are global rejectors (see Section 3). The idea of this rule is that rejection is propagated bottom-up in the HRM.

$$R_4 = \left\{\begin{array}{l} \delta(\texttt{X}, \texttt{Y}, \texttt{M}, \texttt{T}, \texttt{T2})\texttt{:-}\,\varphi(\texttt{X}, \texttt{Y}, \_, \texttt{M}, \texttt{M2}, \texttt{T}), \texttt{satisfied}(\texttt{M2}, \texttt{T}, \texttt{T2}). \\ \delta(\texttt{X}, u^R, \texttt{M}, \texttt{T}, \texttt{T2})\texttt{:-}\,\varphi(\texttt{X}, \_, \_, \texttt{M}, \texttt{M2}, \texttt{T}), \texttt{failed}(\texttt{M2}, \texttt{T}, \texttt{T2}). \end{array}\right\}.$$

The last set, R_5, encodes the accepting/rejecting criteria. Remember that the `last(T)` predicate indicates that T is the last step of a trace. Therefore, the trace is accepted if the root RM is satisfied from the initial step 0 to step T+1 (the step after the last step of the trace, once the final label has been processed). In contrast, the trace is rejected if a rejecting state in the hierarchy is reached between these two same steps.

$$R_5 = \left\{\begin{array}{l} \texttt{accept}\texttt{:-}\,\texttt{last}(\texttt{T}), \texttt{satisfied}(\mathcal{A}_r, \texttt{0}, \texttt{T+1}). \\ \texttt{reject}\texttt{:-}\,\texttt{last}(\texttt{T}), \texttt{failed}(\mathcal{A}_r, \texttt{0}, \texttt{T+1}). \end{array}\right\}$$

Unlike the formalism introduced in Section 3, this encoding does not use stacks, which would be costly to do. Here we know the trace to be processed and, therefore, the RMs can be evaluated bottom-up; that is, we start evaluating the lowest level RMs first on different subtraces, and the result of this evaluation is used in higher level RMs.

We now prove the correctness of the ASP encoding. To do so, we first introduce two definitions and a theorem due to Gelfond and Lifschitz [20] that will help us derive the proof.

Definition 11. Given a label trace λ^*, where $* \in \{G, D, I\}$, an HRM $\mathcal{H}$ is valid with respect to λ^* if $\mathcal{H}$ accepts λ^* and $* = G$ (i.e., λ^* is a goal trace), or $\mathcal{H}$ rejects λ^* and $* = D$ (i.e., λ^* is a dead-end trace), or $\mathcal{H}$ does not accept nor reject λ^* and $* = I$ (i.e., λ^* is an incomplete trace).

Definition 12. An ASP program P is stratified when there is a partition

$$P = P_0 \cup P_1 \cup \cdots \cup P_n \qquad\qquad (P_i \text{ and } P_j \text{ disjoint for all } i \neq j)$$

such that, (1) for every predicate p, the definition of p (all clauses with p in the head) is contained in one of the partitions P_i and, (2) for each $1 \leq i \leq n$, if a predicate occurs positively in a clause of P_i then its definition is contained within $\bigcup_{j \leq i} P_j$, and if a predicate occurs negatively in a clause of P_i then its definition is contained within $\bigcup_{j<i} P_j$.

Theorem 3. *If an ASP program P is stratified, then it has a unique answer set.*

Proposition 1 (Correctness of the ASP encoding)**.** *Given a finite label trace λ^*, where $* \in \{G, D, I\}$, and an HRM $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}\rangle$ that is valid with respect to λ^*, the program $P = M(\mathcal{H}) \cup R \cup M(\lambda^*)$ has a unique answer set AS and (1)* `accept` $\in AS$ *if and only if $* = G$, and (2)* `reject` $\in AS$ *if and only if $* = D$.*

Proof. First, we prove that the program $P = M(\mathcal{H}) \cup R \cup M(\lambda^*)$, where $R = \bigcup_{i=0}^{5} R_i$, has a unique answer set. By Theorem 3, if P is stratified then it has a unique answer set. Therefore, we show there is a possible way of partitioning P following the constraints in Definition 12. A possible partition is $P = P_0 \cup P_1 \cup P_2 \cup P_3$, where $P_0 = M(\lambda^*)$, $P_1 = M(\mathcal{H})$, $P_2 = R_0 \cup R_1$, $P_3 = R_2 \cup R_3 \cup R_4 \cup R_5$. The unique answer set $AS = AS_0 \cup AS_1 \cup AS_2 \cup AS_3$, where AS_i corresponds to partition P_i, is shown below. For simplicity, $\lambda^*[t]$ denotes the t-th label in trace λ^*, $\lambda^*[t:]$ denotes the subtrace starting from the t-th label onwards, and $\mathcal{A}_i(\lambda^*)$ denotes the hierarchy traversal using RM $\mathcal{A}_i$ as the root.

$$AS_0 = \{\texttt{label}(p,t).\mid 0 \le t \le n, p \in L_t\} \cup \{\texttt{step}(t). \mid 0 \le t \le n\} \cup \{\texttt{last}(n).\},$$

$$AS_1 = \begin{array}{l}
\{\texttt{state}(u,\mathcal{A}_i). \mid \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}, u \in U_i\} \cup \\
\left\{ \texttt{call}(u,u',x+e,\mathcal{A}_i,\mathcal{A}_j). \left| \begin{array}{l} \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}, \mathcal{A}_j \in \mathcal{A}, u,u' \in U_i, \varphi_i(u,u',\mathcal{A}_j) \neq \bot, \\ x = \sum_{k=0}^{j-1} |\varphi_i(u,u',\mathcal{A}_k)|, e \in [1, |\varphi_i(u,u',\mathcal{A}_j)|] \end{array} \right.\right\} \cup \\
\left\{ \bar{\varphi}(u,u',x+e,\mathcal{A}_i,t). \left| \begin{array}{l} 0 \le t \le n, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}, \mathcal{A}_j \in \mathcal{A}, u,u' \in U_i, \\ \varphi_i(u,u',\mathcal{A}_j) \neq \bot, x = \sum_{k=0}^{j-1} |\varphi_i(u,u',\mathcal{A}_k)|, \\ e \in [1, |\varphi_i(u,u',\mathcal{A}_j)|], \lambda^*[t] \not\models \varphi_i(u,u',\mathcal{A}_j)[e] \end{array} \right.\right\}
\end{array},$$

$$AS_2 = \begin{array}{l}
\left\{ \varphi(u,u',x+e,\mathcal{A}_i,t). \left| \begin{array}{l} 0 \le t \le n, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}, \mathcal{A}_j \in \mathcal{A}, u,u' \in U_i, \\ \varphi_i(u,u',\mathcal{A}_j) \neq \bot, x = \sum_{k=0}^{j-1} |\varphi_i(u,u',\mathcal{A}_k)|, \\ e \in [1, |\varphi_i(u,u',\mathcal{A}_j)|], \lambda^*[t] \models \varphi_i(u,u',\mathcal{A}_j)[e] \end{array} \right.\right\} \cup \\
\{\texttt{pre\_sat}(u,\mathcal{A}_i,t). \mid 0 \le t \le n, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}, u \in U_i, \lambda^*[t] \models \Phi_{i,u,\top}\}
\end{array},$$

$$AS_3 = \begin{array}{l}
\{\texttt{reachable}(u^0,\mathcal{A}_r,0,0).\} \cup \\
\{\texttt{reachable}(u^0,\mathcal{A}_i,t,t). \mid 0 \le t \le n, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top,\mathcal{A}_r\}, u^0 \in U_i\} \cup \\
\left\{ \texttt{reachable}(u,\mathcal{A}_r,t_1,t_2). \left| \begin{array}{l} 0 \le t_1 < t_2 \le n+1, u \in U_r, \\ \mathcal{H}(\lambda^*[t_1:])[t_2-t_1] = \langle \mathcal{A}_r, u, \cdot, \cdot\rangle \end{array} \right.\right\} \cup \\
\left\{ \texttt{reachable}(u,\mathcal{A}_i,t_1,t_2). \left| \begin{array}{l} 0 \le t_1 < t_2 \le n+1, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_r,\mathcal{A}_\top\}, u \in U_i, \\ \lambda^*[t_1] \models \Phi_{i,u^0,\top}, \\ \mathcal{A}_i(\lambda^*[t_1:])[t_2-t_1] = \langle \mathcal{A}_i, u, \cdot, \cdot\rangle, \\ \mathcal{A}_i(\lambda^*[t_1:])[t_2-t_1-1] \neq \langle \mathcal{A}_i, u^A, \cdot, \cdot\rangle \end{array} \right.\right\} \cup \\
\{\texttt{satisfied}(\mathcal{A}_r,t_1,t_2) \mid 0 \le t_1 < t_2 \le n+1, \mathcal{H}(\lambda^*[t_1:])[t_2-t_1] = \langle \mathcal{A}_r, u^A, \cdot, \cdot\rangle\} \\
\left\{ \texttt{satisfied}(\mathcal{A}_i,t_1,t_2). \left| \begin{array}{l} 0 \le t_1 < t_2 \le n+1, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_r,\mathcal{A}_\top\}, \\ \lambda^*[t_1] \models \Phi_{i,u^0,\top}, \\ \mathcal{A}_i(\lambda^*[t_1:])[t_2-t_1] = \langle \mathcal{A}_i, u^A, \cdot, \cdot\rangle, \\ \mathcal{A}_i(\lambda^*[t_1:])[t_2-t_1-1] \neq \langle \mathcal{A}_i, u^A, \cdot, \cdot\rangle \end{array} \right.\right\} \cup \\
\{\texttt{satisfied}(\mathcal{A}_\top,t,t+1) \mid 0 \le t \le n\} \cup \\
\{\texttt{failed}(\mathcal{A}_r,t_1,t_2) \mid 0 \le t_1 < t_2 \le n+1, \mathcal{H}(\lambda^*[t_1:])[t_2-t_1] = \langle \cdot, u^R, \cdot, \cdot\rangle\} \cup \\
\left\{ \texttt{failed}(\mathcal{A}_i,t_1,t_2). \left| \begin{array}{l} 0 \le t_1 < t_2 \le n+1, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_r,\mathcal{A}_\top\}, \\ \lambda^*[t_1] \models \Phi_{i,u^0,\top}, \\ \mathcal{A}_i(\lambda^*[t_1:])[t_2-t_1] = \langle \cdot, u^R, \cdot, \cdot\rangle \end{array} \right.\right\} \cup \\
\left\{ \delta(u,u',\mathcal{A}_i,t,t+1). \left| \begin{array}{l} 0 \le t \le n, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}, u,u' \in U_i, \\ \lambda^*[t_1] \models \varphi_i(u,u',\mathcal{A}_\top) \end{array} \right.\right\} \cup \\
\left\{ \delta(u,u',\mathcal{A}_i,t_1,t_2). \left| \begin{array}{l} 0 \le t_1 < t_2 \le n+1, \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}, u,u' \in U_i, \\ \exists \mathcal{A}_j \in \mathcal{A}\setminus\{\mathcal{A}_\top\} \text{ s.t. } \phi = \varphi_i(u,u',\mathcal{A}_j), \lambda^*[t_1] \models \Phi_{j,u^0,\phi}, \\ \mathcal{A}_j(\lambda^*[t_1:])[t_2-t_1] = \langle \mathcal{A}_j, u^A, \cdot, \cdot\rangle, \\ \mathcal{A}_j(\lambda^*[t_1:])[t_2-t_1-1] \neq \langle \mathcal{A}_j, u^A, \cdot, \cdot\rangle \end{array} \right.\right\} \cup \\
\left\{ \delta(u,u^R,\mathcal{A}_i,t_1,t_2). \left| \begin{array}{l} \mathcal{A}_i \in \mathcal{A}\setminus\{\mathcal{A}_\top\}, u \in U_i, 0 \le t_1 < t_2 \le n+1, \\ \exists \mathcal{A}_j \in \mathcal{A}\setminus\{\mathcal{A}_\top\} \text{ s.t. } \phi = \varphi_i(u,u',\mathcal{A}_j), \lambda^*[t_1] \models \Phi_{j,u^0,\phi}, \\ \mathcal{A}_j(\lambda^*[t_1:])[t_2-t_1] = \langle \mathcal{A}_k, u^R, \cdot, \cdot\rangle, \mathcal{A}_k \in \mathcal{A} \end{array} \right.\right\} \cup \\
\{\texttt{accept} \mid \mathcal{H}(\lambda^*)[n+1] = \langle \mathcal{A}_r, u^A, \cdot, \cdot\rangle\} \cup \\
\{\texttt{reject} \mid \mathcal{H}(\lambda^*)[n+1] = \langle \mathcal{A}_k, u^R, \cdot, \cdot\rangle, \mathcal{A}_k \in \mathcal{A}\setminus\{\mathcal{A}_\top\}\}
\end{array}.$$

We now prove that $\texttt{accept} \in AS$ if and only if $* = G$ (i.e., the trace achieves the goal). If $* = G$ then, since the hierarchy is valid with respect to λ^* (see Definition 11), the hierarchy traversal $\mathcal{H}(\lambda^*)$ finishes in the accepting state u_A of the root; that is, $\mathcal{H}(\lambda^*)[n+1] = \langle \mathcal{A}_r, u_r^A, \cdot, \cdot \rangle$. This holds if and only if $\texttt{accept} \in AS$.

The proof showing that $\texttt{reject} \in AS$ if and only if $* = D$ (i.e., the trace reaches a dead-end) is similar to the previous one. If $* = D$ then, since the hierarchy is valid with respect to λ^*, the hierarchy traversal $\mathcal{H}(\lambda^*)$ finishes in a rejecting state u_R; that is, $\mathcal{H}(\lambda^*)[n+1] = \langle \mathcal{A}_k, u_R, \cdot, \cdot \rangle$, where $\mathcal{A}_k \in \mathcal{A}$. This holds if and only if $\texttt{reject} \in AS$. □

F.2 Representation of the HRM Learning Task in ILASP

We here formalize the learning of an HRM and its mapping to a general ILASP learning task. We start by defining the HRM learning task introduced in Section 5.

Definition 13. An *HRM learning task* is a tuple $T_{\mathcal{H}} = \langle r, U, \mathcal{P}, \mathcal{A}, \mathcal{A}_{\mathcal{C}}, u^0, u^A, u^R, \Lambda, \kappa \rangle$, where r is the index of the root RM in the HRM; $U \supseteq \{u^0, u^A, u^R\}$ is a set of states of the root RM always containing an initial state u^0, an accepting state u^A, and a rejecting state u^R; $\mathcal{P}$ is a set of propositions; $\mathcal{A} \supseteq \{\mathcal{A}_\top\}$ is a set of RMs; $\mathcal{A}_{\mathcal{C}} \subseteq \mathcal{A}$ is a set of callable RMs; $\Lambda = \Lambda^G \cup \Lambda^D \cup \Lambda^I$ is a set of label traces; and κ is the maximum number of conjunctions/disjuncts in each formula. An HRM $\mathcal{H} = \langle \mathcal{A} \cup \{\mathcal{A}_r\}, \mathcal{A}_r, \mathcal{P} \rangle$ is a solution of $T_{\mathcal{H}}$ if and only if it is valid with respect to all the traces in Λ.

We make some assumptions about the sets of RMs $\mathcal{A}$: (i) all RMs reachable from RMs in $\mathcal{A}_{\mathcal{C}}$ must be in $\mathcal{A}$, (ii) all RMs in $\mathcal{A}$ are deterministic, and (iii) all RMs in $\mathcal{A}$ are defined over the same set of propositions $\mathcal{P}$ (or a subset of it).

For completeness, we provide the definition of an ILASP task introduced by Law et al. [31]. The first definition corresponds to the form of the examples taken by ILASP, while the second corresponds to the ILASP tasks themselves.

Definition 14. A *context-dependent partial interpretation* (CDPI) is a pair $\langle \langle e^{inc}, e^{exc} \rangle, e^{ctx} \rangle$, where $\langle e^{inc}, e^{exc} \rangle$ is a pair of sets of atoms, called a *partial interpretation*, and e^{ctx} is an ASP program called a *context*. A program P *accepts* a CDPI $\langle \langle e^{inc}, e^{exc} \rangle, e^{ctx} \rangle$ if and only if there is an answer set A of $P \cup e^{ctx}$ such that $e^{inc} \subseteq A$ and $e^{exc} \cap A = \emptyset$.

Definition 15. An *ILASP task* is a tuple $T = \langle B, S_M, \langle E^+, E^- \rangle \rangle$ where B is the ASP background knowledge, which describes a set of known concepts before learning; S_M is the set of ASP rules allowed in the hypotheses; and E^+ and E^- are sets of CDPIs called, respectively, the positive and negative examples. A hypothesis $H \subseteq S_M$ is an *inductive solution* of T if and only if (i) $\forall e \in E^+$, $B \cup H$ accepts e, and (ii) $\forall e \in E^-$, $B \cup H$ does not accept e.

Given an HRM learning task $T_{\mathcal{H}}$, we map it into an ILASP learning task $M(T_{\mathcal{H}}) = \langle B, S_M, \langle E^+, \emptyset \rangle \rangle$ and use the ILASP system [30] to find an inductive solution $M_\varphi(\mathcal{H}) \subseteq S_M$ that covers the examples. Note that we do not use *negative examples* ($E^- = \emptyset$). We define the components of $M(T_{\mathcal{H}})$ below.

Background Knowledge. The background knowledge $B = B_U \cup B_{\mathcal{A}} \cup R$ is a set of rules that describe the behavior of the HRM. The set B_U consists of $\texttt{state}(u, \mathcal{A}_r)$ facts for each state $u \in U$ of the root RM with index r we aim to induce, whereas $B_{\mathcal{A}} = \bigcup_{\mathcal{A}_i \in \mathcal{A} \setminus \{\mathcal{A}_\top\}} M(\mathcal{A}_i)$ contains the ASP representations of all RMs. Finally, R is the set of general rules introduced in Appendix F.1 that defines how HRMs process label traces. Importantly, the index of the root r in these rules must correspond to the one used in $T_{\mathcal{H}}$.

Hypothesis Space. The hypothesis space S_M contains all $\texttt{ed}$ and $\bar{\varphi}$ rules that characterize a transition from a non-terminal state $u \in U \setminus \{u^A, u^R\}$ to a different state $u' \in U \setminus \{u\}$ using edge $i \in [1, \kappa]$. Formally, it is defined as

$$S_M = \left\{ \begin{array}{l|c} \texttt{call}(u, u', i, \mathcal{A}_m). & u \in U \setminus \left\{u^A, u^R\right\}, \\ \bar{\varphi}(u, u', i, \mathcal{A}_m, \texttt{T})\,\texttt{:-}\,\texttt{label}(p, \texttt{T}), \texttt{step}(\texttt{T}). & u' \in U \setminus \{u\}, i \in [1, \kappa], \\ \bar{\varphi}(u, u', i, \mathcal{A}_m, \texttt{T})\,\texttt{:-}\,\texttt{not label}(p, \texttt{T}), \texttt{step}(\texttt{T}). & \mathcal{A}_m \in \mathcal{A}_{\mathcal{C}}, p \in \mathcal{P} \end{array} \right\}.$$

Example Sets. Given a set of traces $\Lambda = \Lambda^G \cup \Lambda^D \cup \Lambda^I$, the set of *positive examples* is defined as

$$E^+ = \{\langle e^*, M(\lambda) \rangle \mid * \in \{G, D, I\}, \lambda \in \Lambda^*\},$$

where

- $e^G = \langle \{\texttt{accept}\}, \{\texttt{reject}\} \rangle$,
- $e^D = \langle \{\texttt{reject}\}, \{\texttt{accept}\} \rangle$, and
- $e^I = \langle \{\}, \{\texttt{accept}, \texttt{reject}\} \rangle$

are the partial interpretations for goal, dead-end and incomplete traces. The `accept` and `reject` atoms express whether a trace is accepted or rejected by the HRM; hence, goal traces must only be accepted, dead-end traces must only be rejected, and incomplete traces cannot be accepted or rejected. Note that the context of each example is the set of ASP facts $M(\lambda)$ that represents the corresponding trace (see Definition 9).

Correctness of the Learning Task. The following theorem captures the correctness of the HRM learning task.

Theorem 4. *Given an HRM learning task $T_{\mathcal{H}} = \langle r, U, \mathcal{P}, \mathcal{A}, \mathcal{A}_{\mathcal{C}}, u_0, u_A, u_R, \Lambda, \kappa \rangle$, an HRM $\mathcal{H} = \langle \mathcal{A} \cup \{\mathcal{A}_r\}, \mathcal{A}_r, \mathcal{P} \rangle$ is a solution of $T_{\mathcal{H}}$ if and only if $M_{\varphi}(\mathcal{A}_r)$ is an inductive solution of $M(T_{\mathcal{H}}) = \langle B, S_M, \langle E^+, \emptyset \rangle \rangle$.*

Proof. Assume $\mathcal{H}$ is a solution of $T_{\mathcal{H}}$.

$\iff$ $\mathcal{H}$ is valid with respect to all traces in Λ (i.e., $\mathcal{H}$ accepts all traces in Λ^G, rejects all traces in Λ^D and does not accept nor reject any trace in Λ^I).

$\iff$ By Proposition 1, for each trace $\lambda^* \in \Lambda^*$ where $* \in \{G, D, I\}$, $M(\mathcal{H}) \cup R \cup M(\lambda^*)$ has a unique answer set AS and (1) $\texttt{accept} \in AS$ if and only if $* = G$, and (2) $\texttt{reject} \in AS$ if and only if $* = D$.

$\iff$ For each example $e \in E^+$, $R \cup M(\mathcal{H})$ accepts e.

$\iff$ For each example $e \in E^+$, $B \cup M_{\varphi}(\mathcal{A}_r)$ accepts e (the two programs are identical).

$\iff$ $M_{\varphi}(\mathcal{A}_r)$ is an inductive solution of $M(T_{\mathcal{H}})$. □

Constraints. We introduce several constraints encoding structural properties of the HRMs we want to learn. Some of these constraints are expressed in terms of facts $\texttt{pos}(u, u', e, m, p)$ and $\texttt{neg}(u, u', e, m, p)$, which indicate that proposition $p \in \mathcal{P}$ appears positively (resp. negatively) in edge e from state u to state u' in RM $\mathcal{A}_m$. These facts are derived from $\bar{\varphi}$ rules in $M(\mathcal{H})$ and injected in the ILASP tasks using meta-program injection [32].

The following set of constraints ensures that the learned root RM is *deterministic* using the *saturation* technique [16]. The idea is to check determinism top-down by selecting two edges from a given state in the root, each associated with a set of literals. Initially, the set of literals is formed by those in the formula labeling the edges. If a selected edge calls a non-leaf RM, we select an edge from the initial state of the called RM, augment the set of literals with the associated formula, and repeat the process until a call to the leaf RM is reached. We then check if the literal sets are mutually exclusive. If there is a pair of edges from the root that are not mutually exclusive, the solution is discarded. The set of rules is shown below. The first rule states that we keep two saturation IDs, one for each of the edges we select next and for which mutual exclusivity is checked. The second rule chooses a state `X` in the root, whereas the third rule selects two edges from this state and assigns a saturation ID to each of them. The fourth rule indicates that if one of the edges we have selected so far calls a non-leaf RM, we select one of the edges from the initial state of the called RM and create a new edge with the same saturation ID. The fifth and sixth rules take the propositions for each set of edges (one per saturation ID). The next three rules indicate that if the edges are mutually exclusive (i.e., a proposition appears positively in one set and negatively in the other) or they are the same, then the answer set is saturated. The saturation itself is encoded in the following three rules: an answer set is saturated by adding every possible `ed_mtx` and `root_point` atoms to the answer set. Due to the minimality of answer sets in disjunctive answer set programming, this “maximal” interpretation can only be an answer set if there is no smaller answer set. This will be the case if and only if every choice of edges satisfies the condition (i.e. every choice of `ed_mtx` and `root_point` atoms results in saturation). The constraint encoded in the final rule then discards answer sets in which saturation did not occur, meaning that the remaining solutions must satisfy the condition.

$$\left\{\begin{array}{l}
\texttt{sat\_id(1;2).}\\
\texttt{root\_point(X,M) : call(X,\_,\_,M,\_), M=}\mathcal{A}_r\texttt{.}\\
\texttt{ed\_mtx((X,Y,E,M,M2),SatID) : call(X,Y,E,M,M2) :- root\_point(X,M), sat\_id(SatID).}\\
\texttt{ed\_mtx((u0,Y2,E2,M2,M3),SatID) : call(u0,Y2,E2,M2,M3) :- ed\_mtx((\_,\_,\_,\_,M2),SatID),}\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\texttt{M2!=}\mathcal{A}_\top\texttt{.}\\
\texttt{pos\_prop(P,ID) :- ed\_mtx((X,Y,E,M,\_),ID), pos(X,Y,E,M,P).}\\
\texttt{neg\_prop(P,ID) :- ed\_mtx((X,Y,E,M,\_),ID), neg(X,Y,E,M,P).}\\
\texttt{saturate :- pos\_prop(P,1), neg\_prop(P,2).}\\
\texttt{saturate :- pos\_prop(P,2), neg\_prop(P,1).}\\
\texttt{saturate :- ed\_mtx((X,Y,\_,M,M2),1), ed\_mtx((X,Y,\_,M,M2),2), root\_point(X,M).}\\
\texttt{root\_point(X,M) :- call(X,\_,\_,M,\_), saturate, M=}\mathcal{A}_r\texttt{.}\\
\texttt{ed\_mtx((X,Y,E,M,M2),SatID) :- call(X,Y,E,M,M2), M=}\mathcal{A}_r\texttt{, sat\_id(SatID), saturate.}\\
\texttt{ed\_mtx((u0,Y,E,M,M2),SatID) :- call(u0,Y,E,M,M2), sat\_id(SatID), saturate.}\\
\texttt{:- not saturate.}
\end{array}\right\}$$

Other required constraints to learn sensible HRMs are shown below. The first rule prevents an edge from being labeled with calls to two different RMs. The second rule prevents edges from being labeled with the same literal both positively and negatively.

$$\left\{\begin{array}{l}
\texttt{:- call(X,Y,E,M,M2), call(X,Y,E,M,M3), M2!=M3.}\\
\texttt{:- pos(X,Y,E,M,P), neg(X,Y,E,M,P).}
\end{array}\right\}$$

The following constraints are used to speed up the learning of an HRM. First, we extend the *symmetry breaking* method by Furelos-Blanco et al. [17], originally proposed for flat RMs, to our hierarchical setting. The main advantage of this method is that it accelerates learning without restricting the family of learnable HRMs. Other constraints analogous to those in previous work [17] that speed up the learning process further are enumerated below. For simplicity, some of these constraints use the auxiliary rule below to define the $\texttt{ed(X,Y,E,M)}$ predicate, which is equivalent to the $\texttt{call(X,Y,E,M,M2)}$ predicate but omitting the called RM:

$$\texttt{ed(X,Y,E,M) :- call(X,Y,E,M,\_).}$$

The constraints are the following:

- Rule out inductive solutions where an edge calling the leaf $\mathcal{A}_\top$ is labeled by a formula formed only by negated propositions. The rule below enforces a proposition to occur positively whenever a proposition appears negatively in an edge calling $\mathcal{A}_\top$.

$$\texttt{:- neg(X,Y,E,M,\_), not pos(X,Y,E,M,\_), call(X,Y,E,M,}\mathcal{A}_\top\texttt{).}$$

- Rule out any inductive solution where an edge from X to Y with index E is not labeled by a positive or a negative literal. This rule only applies to calls to the leaf $\mathcal{A}_\top$, thus avoiding unconditional transitions.

$$\texttt{:- not pos(X,Y,E,M,\_), not neg(X,Y,E,M,\_), call(X,Y,E,M,}\mathcal{A}_\top\texttt{).}$$

- Rule out inductive solutions containing states different from the accepting and rejecting states without outgoing edges. In general, these states are not interesting.

$$\left\{\begin{array}{l}
\texttt{has\_outgoing\_edges(X,M) :- ed(X,\_,\_,M).}\\
\texttt{:- state(X,M), not has\_outgoing\_edges(X,M), X!=}u^A\texttt{, X!=}u^R\texttt{.}
\end{array}\right\}$$

- Rule out inductive solutions containing cycles; that is, solutions where two states can be reached from each other. The $\texttt{path(X,Y,M)}$ predicate indicates there is a directed path (i.e., a sequence of directed edges) from X to Y in RM M. The first rule states that there is a path from X to Y if there is an edge from X to Y. The second rule indicates that there is a path from X to Y if there is an edge from X to an intermediate state Z from which there is a path to Y. Finally, the third rule discards the solutions where X and Y can be reached from each other through directed edges.

$$\left\{\begin{array}{l}
\texttt{path(X,Y,M) :- ed(X,Y,\_,M).}\\
\texttt{path(X,Y,M) :- ed(X,Z,\_,M), path(Z,Y,M).}\\
\texttt{:- path(X,Y,M), path(Y,X,M).}
\end{array}\right\}$$

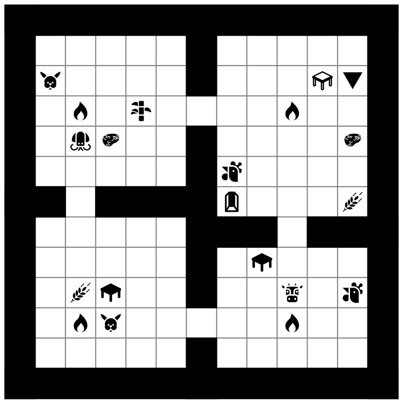


Figure 8: An instance of the CRAFTWORLD grid in the FRL setting.

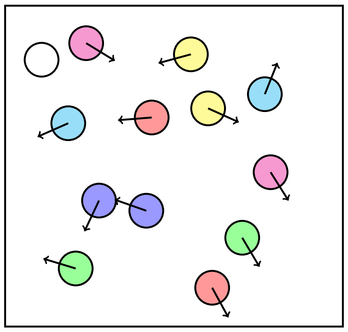


Figure 9: An instance of the WATERWORLD in the WOD setting [49].

G Experimental Details

In this section, we describe the details of the experiments introduced in Section 6. We discuss how the domains are implemented, the hyperparameters used to run the algorithms, and provide all specific results through tables and plots. All experiments ran on 3.40GHz Intel® Core™ i7-6700 processors.

G.1 Domains

The CRAFTWORLD domain is based on MiniGrid [11], thus inheriting many of its features. At each step, the agent observes a $W \times H \times 3$ tensor, where W and H are the width and height of the grid. The three channels contain the object IDs, the color IDs, and object state IDs (including the orientation of the agent) respectively. Each of the objects we define (except for the lava 🔥, which already existed in MiniGrid) has its own object and color IDs. Before providing the agent with the state, the content of all matrices is scaled between -1 and 1. Note that even though the agent gets a full view of the grid, it is still unaware of the completion degree of a task. Other works have previously used the full view of the grid [23, 24].

The grids are randomly generated. In all settings (OP, OPL, FR, FRL), the agent and the objects are randomly assigned an unoccupied position. In the case of FR and FRL, no object occupies a position between rooms nor its adjoining positions. There is a single object per object type (i.e., proposition) in OP and OPL, whereas there can be one or two per type in FR and FRL. Finally, there is a single lava location in OPL, which is randomly assigned (like the rest of the propositions), whereas in FRL there are four fixed lava locations placed in the intersections between doors as shown in Figure 8.

The WATERWORLD domain (cf. Figure 9) has a continuous state space. The states are vectors containing the absolute position and velocity of the agent, and the relative positions and velocities of the other balls. The agent does not know the color of each ball. In all settings (WOD and WD), a WATERWORLD instance is created by assigning a random position and direction to each ball. Like in CRAFTWORLD, the agent does not know the degree of completion of a task.

G.2 Hyperparameters

Table 2 lists some of the hyperparameters used in the experiments. The rest of the hyperparameters and other details (e.g., architectures, evaluation of other methods) are discussed in the following paragraphs.

Table 2: List of hyperparameters and their values.

Parameter	CraftWorld	WaterWorld
General		
Episodes	300,000	300,000
Maximum episode length	1,000	1,000
Num. of instances I	10	10
DQNs		
Learning rate α	5×10^{-4}	1×10^{-5}
Learning rate (SMDP) α	5×10^{-4}	1×10^{-3}
Optimizer	RMSprop [22]	RMSprop [22]
Discount γ	0.9	0.9
Discount (SMDP) γ	0.99	0.99
Updated formula Q-functions per step	4	4
Replay memory size	500,000	500,000
Replay start size	100,000	100,000
Target network update frequency	1,500	1,500
Replay memory size (SMDP)	10,000	10,000
Replay start size (SMDP)	1,000	1,000
Target network update frequency (SMDP)	500	500
Minibatch size	32	32
Exploration		
Initial exploration	1.0	1.0
Final exploration	0.1	0.1
Annealing steps	2,000,000	5,000,000
Annealing steps (SMDP)	10,000	10,000
HRM learning		
Curriculum weight β	0.99	0.99
Curriculum threshold Δ	0.85	0.75
ILASP time budget	2 hours	2 hours
Num. collected goal traces ρ (height 1)	25	25
Num. collected goal traces ρ (height $\geq$2)	150	150
Num. goal traces ρ_s to learn first HRM	10	10

Architectures. The DQNs for CraftWorld consist of a 3-layer convolutional neural network (CNN) with 16, 32 and 32 filters respectively. All kernels are 2×2 and use a stride of 1. In the FR and FRL settings, there is a max pooling layer with kernel size 2×2 after the first convolutional layer. This part of the architecture is based on that by Igl et al. [23] and Jiang et al. [24], who also work on MiniGrid using the full view of the grid. In DQNs associated with formulas, the CNN's output is fed to a 3-layer multilayer perceptron (MLP) where the hidden layer has 256 rectifier units and the output layer has a single output for each action. In the case of DQNs for RMs, the output of the CNN is extended with the encoding of the RM state and the context (as discussed in Appendix D) before being fed to a 3-layer MLP where the hidden layer has 256 rectifier units and the output layer has a single output for each call in the RM.

The architecture for WaterWorld is a simple modification of the one introduced by Toro Icarte et al. [49]. The formula DQNs consist of a 5-layer MLP, where each of the 3 hidden layers has 512 rectifier units. The DQN for the RMs share the same architecture and, like in CraftWorld, the state from the environment is extended with the state and context encodings.

Compression. Akin to some methods for learning RMs [17, 50], we compress label traces by merging consecutive equal labels into a single one. For instance, $\langle\{\}, \{$🪣$\}, \{$🪣$\}, \{\}, \{\}, \{$🪑$\}, \{$🐮$\}\rangle$ becomes $\langle\{\}, \{$🪣$\}, \{\}, \{$🪑$\}, \{$🐮$\}\rangle$.

Curriculum. The average undiscounted returns (see Section 5 and Appendix E) are updated for each task-instance pair every 100 training episodes using the undiscounted return obtained by the greedy policies in a single evaluation episode.

Flat HRM Learning Methods. We briefly describe the methods used to learn flat HRMs. Each run consists of 150,000 episodes, and the set of instances is exactly the same across methods. The core difference with respect to learning non-flat HRMs is that there is a single task for which the HRM is learned. Our method, LHRM, is therefore not able to reuse previously learned HRMs for other tasks; however, it still uses the same hyperparameters. In the case of DeepSynth [21], LRM [50] and JIRP [55], we exclusively evaluate the RM learning method using traces collected through random walks. For a fair comparison against LHRM (both in the non-flat and flat learning cases), we (i) compress the traces using the aforementioned methodology, and (ii) use the OP and WOD settings of CRAFTWORLD and WATERWORLD respectively, where observing goal traces is very easy for simple tasks such as MILKBUCKET. In these approaches, a different instance is selected at each episode following a cyclic order (i.e., 1, 2,..., $I-1$, I, 1, 2, ...). The set of propositions in these approaches includes a proposition covering the case where no other propositions are observed (if needed). In the case of LRM, one of the parameters is the maximum number of RM states, which we set to that of the minimal RM. Finally, we modify DeepSynth to avoid periodically calling the learner (i.e., only call it when a counterexample trace is observed): this is not done in other approaches and usually causes timeouts unnecessarily (the same RM is repeatedly learned).[5]

ILASP. We use ILASP2 [30] to learn the HRMs.[6] For efficiency, the default calls to the underlying ASP solver are modified to be made with the flag `--opt-mode=ignore`, meaning that non-minimal solutions might be obtained (i.e., solutions involving more rules than needed), so the learned root might contain some unnecessary edges. In practice, the solutions produced by ILASP rarely contain such edges and, if they do, these edges eventually disappear by observing an appropriate counterexample. We hypothesize that using this flag helps since no optimization is made every time ILASP is called. The design of the ILASP tasks is discussed in Appendix F.2. We highlight that this notion of minimality is not related to that of a minimal RM (i.e., an RM with the fewest number of states) described in Section 5.

G.3 Extended Results

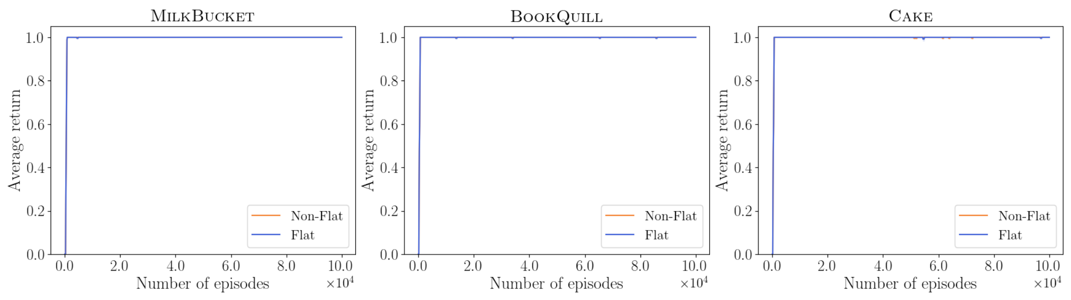
We organize the tables and figures following the structure in Section 6.

Policy Learning. Figure 10 shows the plots omitted in the main paper (the remaining setting, FRL, is shown in Figure 2). The number of plotted episodes varies across domains and tasks for clarity.

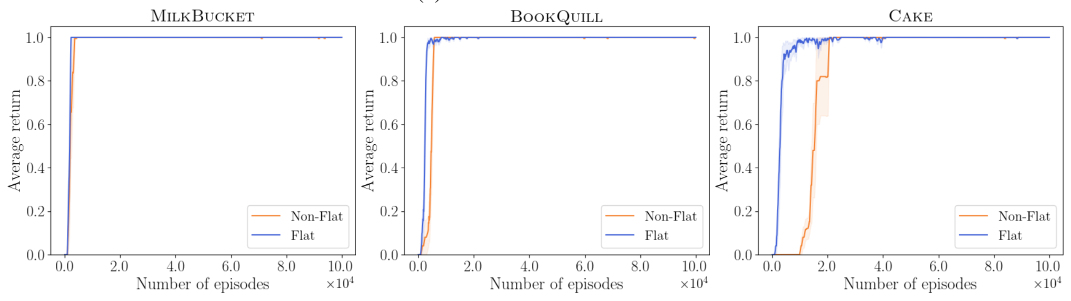
Learning Non-Flat HRMs. We present tables containing the results for the HRM learning component of LHRM. The content of the columns is the following left-to-right: (1) the task; (2) the number of runs in which at least one goal trace was observed; (3) the number of runs in which at least one HRM was learned; (4) time spent to learn the HRMs; (5) number of calls to ILASP made to learn the HRMs; (6) number of states of the final HRM; (7) number of edges of the final HRM; (8) number of episodes between the learning of the first HRM and the activation of the task's level; (9) number of example traces of a given type (G = goal, D = dead-end, I = incomplete); and (10) length of the example traces of a given type. In addition, the bottom of the tables contains the number of completed runs (i.e., the number of runs that have not timed out), the total time spent on learning the HRMs, and the total number of calls made to ILASP. In the case of CRAFTWORLD, Table 3 shows the results for the default case (all lower level RMs are callable and options are used for exploration), Table 4 shows the results when the set of callable RMs contains only those actually needed, and Table 5 shows the results using primitive actions for exploration instead of options. Analogous results are shown for WATERWORLD in Tables 6, 7 and 8. The discussion of these results can be found in Section 6.

[5]The code of these methods is publicly available: DeepSynth (`https://github.com/grockious/deepsynth`, MIT license), LRM (`https://bitbucket.org/RToroIcarte/lrm`, no license), and JIRP (`https://github.com/corazza/stochastic-reward-machines`, no license). The first two links can be found in the papers, whereas the last one was provided by one of the authors through personal communication.

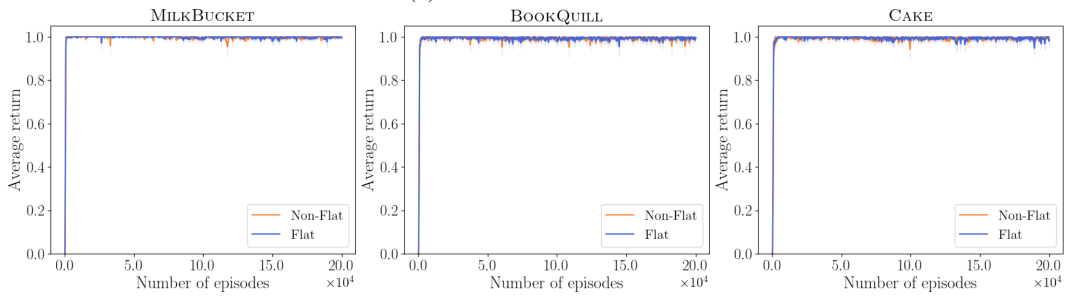
[6]ILASP is free to use for research and education (see `https://www.ilasp.com/terms`).



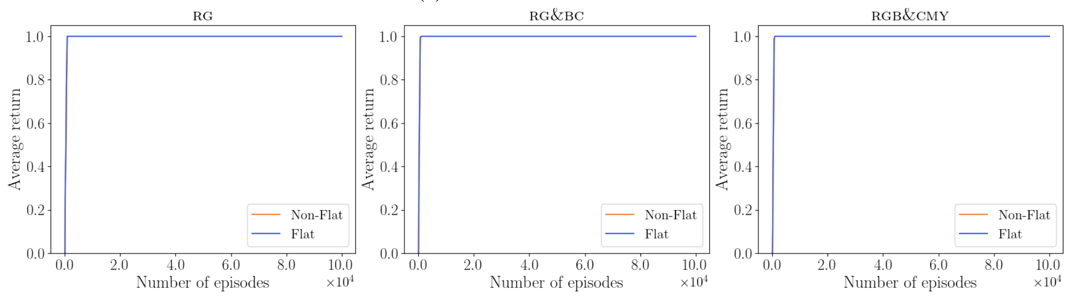
(a) CRAFTWORLD – OP



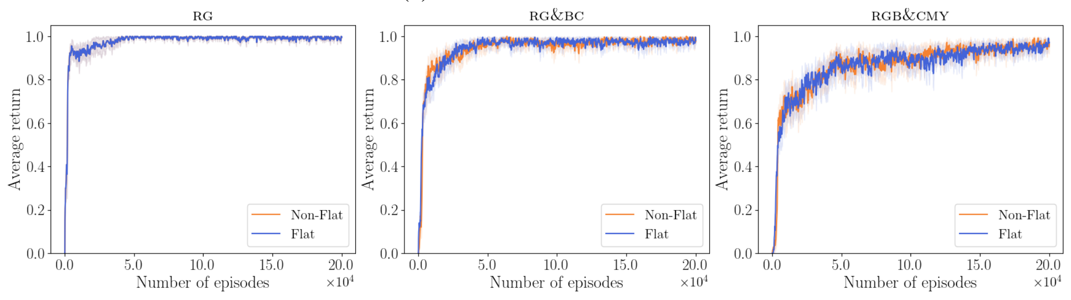
(b) CRAFTWORLD – OPL



(c) CRAFTWORLD – FR



(d) WATERWORLD – WOD



(e) WATERWORLD – WD

Figure 10: Learning curves comparing the performance of handcrafted non-flat and flat HRMs.

Table 3: Results of LHRM in CraftWorld for the default case.

(a) OP

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples		Example Length	
							($\times 10^2$)	G	I	G	I
Batter	5	5	11.1 (1.7)	17.8 (1.9)	5.0 (0.0)	5.2 (0.2)	1.8 (0.1)	12.2 (0.7)	11.6 (1.4)	26.5 (2.1)	24.2 (3.2)
Bucket	5	5	0.9 (0.0)	3.6 (0.2)	3.0 (0.0)	2.0 (0.0)	1.7 (0.1)	10.0 (0.0)	1.6 (0.2)	19.4 (1.1)	19.3 (5.7)
Compass	5	5	135.4 (73.3)	18.6 (1.6)	5.0 (0.0)	5.2 (0.2)	1.8 (0.2)	11.8 (0.6)	12.8 (1.4)	28.7 (1.9)	20.3 (2.8)
Leather	5	5	0.9 (0.0)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	1.8 (0.1)	10.0 (0.0)	1.8 (0.2)	16.7 (1.7)	17.9 (4.4)
Paper	5	5	0.8 (0.1)	3.4 (0.2)	3.0 (0.0)	2.0 (0.0)	1.6 (0.1)	10.0 (0.0)	1.4 (0.2)	19.8 (2.0)	40.6 (27.0)
Quill	5	5	18.0 (3.5)	19.8 (1.2)	5.0 (0.0)	5.2 (0.2)	2.1 (0.1)	13.2 (0.4)	12.6 (1.1)	29.6 (2.5)	24.4 (3.2)
Sugar	5	5	0.8 (0.1)	3.2 (0.2)	3.0 (0.0)	2.0 (0.0)	1.7 (0.2)	10.0 (0.0)	1.2 (0.2)	17.7 (1.6)	17.5 (3.2)
Book	5	5	191.2 (36.4)	22.8 (2.6)	5.0 (0.0)	5.8 (0.2)	6.0 (0.2)	11.4 (0.7)	17.4 (2.2)	20.5 (1.8)	24.8 (1.5)
Map	5	5	549.4 (149.5)	33.4 (3.2)	5.0 (0.0)	5.6 (0.2)	6.0 (0.2)	12.2 (0.6)	27.2 (2.9)	29.5 (3.2)	28.7 (1.7)
MilkBucket	5	5	1.5 (0.2)	4.6 (0.4)	3.0 (0.0)	2.0 (0.0)	6.8 (0.5)	10.0 (0.0)	2.6 (0.4)	11.6 (0.7)	15.3 (4.3)
BookQuill	5	5	17.9 (1.4)	19.6 (1.1)	4.0 (0.0)	4.0 (0.0)	3.8 (0.1)	10.0 (0.0)	16.6 (1.1)	27.2 (1.3)	20.8 (1.4)
MilkB.Sugar	5	5	7.3 (1.2)	12.4 (1.2)	4.0 (0.0)	4.0 (0.0)	3.8 (0.1)	10.2 (0.2)	9.2 (1.2)	16.9 (0.8)	14.3 (1.7)
Cake	5	5	74.5 (25.7)	26.4 (3.7)	4.0 (0.0)	3.2 (0.2)	2.1 (0.1)	10.2 (0.2)	23.2 (3.6)	38.4 (0.9)	22.7 (1.6)
Completed Runs		5									
Total Time (s.)		1009.8 (122.3)									
Total Calls		189.4 (4.1)									

(b) OPL

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							($\times 10^2$)	G	D	I	G	D	I
Batter	5	5	13.7 (2.9)	23.0 (3.0)	6.0 (0.0)	9.2 (0.2)	12.0 (1.0)	11.4 (0.4)	7.0 (1.2)	10.6 (1.6)	20.4 (1.1)	18.7 (1.6)	12.1 (1.7)
Bucket	5	5	1.8 (0.2)	7.2 (0.6)	4.0 (0.0)	4.0 (0.0)	8.0 (0.5)	10.2 (0.2)	2.2 (0.2)	2.8 (0.4)	10.2 (0.5)	13.4 (1.9)	6.8 (1.7)
Compass	5	5	13.1 (1.7)	22.0 (1.7)	6.0 (0.0)	9.2 (0.2)	10.4 (1.4)	11.0 (0.6)	6.8 (1.0)	10.2 (1.0)	17.2 (1.6)	20.9 (1.9)	14.3 (0.8)
Leather	5	5	1.9 (0.2)	7.0 (0.5)	4.0 (0.0)	4.0 (0.0)	6.9 (0.5)	10.0 (0.0)	2.4 (0.2)	2.6 (0.4)	11.1 (0.9)	16.9 (5.6)	8.9 (3.3)
Paper	5	5	2.0 (0.2)	7.6 (0.6)	4.0 (0.0)	4.0 (0.0)	7.7 (1.1)	10.0 (0.0)	3.0 (0.3)	2.6 (0.4)	10.1 (0.9)	18.9 (3.3)	5.6 (0.8)
Quill	5	5	11.3 (1.2)	22.0 (1.2)	6.0 (0.0)	9.2 (0.2)	12.8 (1.5)	10.6 (0.2)	6.4 (0.7)	11.0 (0.9)	15.3 (1.3)	13.5 (1.0)	12.1 (1.4)
Sugar	5	5	1.7 (0.1)	6.4 (0.4)	4.0 (0.0)	4.0 (0.0)	6.5 (0.7)	10.0 (0.0)	2.4 (0.2)	2.0 (0.3)	9.6 (0.6)	15.3 (3.6)	16.6 (9.2)
Book	5	5	427.8 (201.6)	32.6 (4.2)	6.0 (0.0)	6.6 (0.2)	5.6 (0.2)	12.0 (0.3)	3.6 (0.7)	23.0 (3.4)	21.6 (1.5)	25.9 (3.4)	23.7 (1.3)
Map	5	5	647.9 (110.7)	38.6 (3.6)	6.0 (0.0)	6.4 (0.2)	5.6 (0.2)	11.2 (0.4)	3.8 (0.9)	29.6 (3.5)	23.1 (1.0)	27.8 (4.6)	26.1 (0.4)
MilkBucket	5	5	2.1 (0.2)	5.4 (0.4)	4.0 (0.0)	3.0 (0.0)	7.6 (0.5)	10.0 (0.0)	1.4 (0.4)	2.0 (0.0)	11.1 (0.5)	26.3 (6.5)	15.2 (5.8)
BookQuill	5	5	18.7 (2.3)	16.6 (1.3)	4.0 (0.0)	4.0 (0.0)	3.7 (0.2)	10.0 (0.0)	0.4 (0.2)	13.2 (1.4)	29.0 (1.1)	6.2 (5.5)	27.8 (1.4)
MilkB.Sugar	5	5	7.7 (0.7)	12.2 (0.9)	4.0 (0.0)	4.0 (0.0)	3.8 (0.2)	10.0 (0.0)	0.2 (0.2)	9.0 (0.9)	16.0 (0.9)	1.6 (1.6)	16.3 (1.3)
Cake	5	5	472.9 (216.6)	36.0 (6.0)	5.0 (0.0)	4.6 (0.2)	2.1 (0.0)	10.0 (0.0)	1.6 (0.4)	31.4 (5.7)	39.5 (1.2)	41.5 (8.6)	26.9 (0.8)
Completed Runs		5											
Total Time (s.)		1622.6 (328.7)											
Total Calls		236.6 (9.3)											

(c) FR

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples		Example Length	
							($\times 10^2$)	G	I	G	I
Batter	5	5	12.3 (1.7)	17.6 (1.3)	5.0 (0.0)	5.4 (0.2)	9.2 (1.2)	11.6 (0.4)	12.0 (1.2)	30.3 (2.3)	27.8 (2.0)
Bucket	5	5	1.2 (0.1)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	6.7 (0.9)	10.0 (0.0)	1.8 (0.2)	16.6 (2.5)	28.5 (4.1)
Compass	5	5	14.1 (1.6)	20.2 (1.7)	5.0 (0.0)	5.2 (0.2)	9.8 (0.7)	11.6 (0.6)	14.6 (1.2)	26.5 (0.8)	26.5 (2.1)
Leather	5	5	1.1 (0.1)	3.6 (0.2)	3.0 (0.0)	2.0 (0.0)	4.5 (0.7)	10.0 (0.0)	1.6 (0.2)	13.4 (1.3)	16.7 (3.6)
Paper	5	5	1.2 (0.0)	4.0 (0.0)	3.0 (0.0)	2.0 (0.0)	4.9 (0.9)	10.0 (0.0)	2.0 (0.0)	12.4 (1.1)	10.9 (2.5)
Quill	5	5	8.9 (0.9)	16.0 (0.8)	5.0 (0.0)	5.2 (0.2)	9.4 (1.7)	10.6 (0.2)	11.4 (0.6)	25.4 (0.3)	25.5 (2.7)
Sugar	5	5	1.1 (0.1)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	5.2 (0.3)	10.0 (0.0)	1.8 (0.2)	15.3 (1.7)	21.0 (10.1)
Book	5	5	220.2 (83.3)	25.2 (3.4)	5.0 (0.0)	5.6 (0.2)	6.1 (0.2)	10.2 (0.2)	21.0 (3.4)	21.9 (1.0)	18.4 (0.7)
Map	5	5	628.3 (85.4)	37.8 (3.7)	5.0 (0.0)	5.6 (0.2)	5.8 (0.1)	10.0 (0.0)	33.8 (3.7)	26.4 (1.0)	21.4 (0.7)
MilkBucket	5	5	1.9 (0.2)	5.0 (0.3)	3.0 (0.0)	2.0 (0.0)	9.8 (0.7)	10.0 (0.0)	3.0 (0.3)	13.2 (0.7)	12.8 (3.2)
BookQuill	5	5	12.9 (2.2)	15.6 (1.7)	4.0 (0.0)	4.0 (0.0)	3.9 (0.1)	10.0 (0.0)	12.6 (1.7)	29.0 (1.5)	13.3 (0.8)
MilkB.Sugar	5	5	7.2 (0.6)	12.0 (0.7)	4.0 (0.0)	4.0 (0.0)	3.9 (0.2)	10.0 (0.0)	9.0 (0.7)	18.9 (0.9)	10.1 (1.0)
Cake	5	5	121.1 (41.1)	34.0 (4.8)	4.0 (0.0)	3.0 (0.0)	2.2 (0.0)	10.0 (0.0)	31.0 (4.8)	42.2 (1.7)	16.2 (1.1)
Completed Runs		5									
Total Time (s.)		1031.6 (150.3)									
Total Calls		198.6 (11.3)									

(d) FRL

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							($\times 10^2$)	G	D	I	G	D	I
Batter	5	5	11.3 (1.4)	23.4 (2.5)	6.0 (0.0)	9.2 (0.2)	468.4 (121.9)	10.4 (0.2)	7.6 (0.9)	11.4 (1.9)	11.9 (0.6)	10.1 (1.3)	9.9 (0.4)
Bucket	5	5	2.3 (0.2)	7.0 (0.3)	4.0 (0.0)	4.0 (0.0)	129.5 (69.4)	10.2 (0.2)	2.8 (0.2)	2.0 (0.3)	7.8 (0.5)	9.9 (1.7)	6.4 (2.1)
Compass	5	5	13.0 (1.9)	24.6 (2.2)	6.0 (0.0)	9.4 (0.2)	550.8 (156.4)	10.4 (0.2)	7.8 (1.0)	12.4 (1.2)	12.5 (1.6)	9.4 (1.0)	8.4 (0.5)
Leather	5	5	2.5 (0.3)	7.8 (0.7)	4.0 (0.0)	4.0 (0.0)	89.0 (18.0)	10.0 (0.0)	3.2 (0.4)	2.6 (0.4)	7.3 (0.4)	9.3 (1.7)	3.7 (0.4)
Paper	5	5	2.2 (0.1)	7.0 (0.3)	4.0 (0.0)	4.0 (0.0)	82.7 (18.8)	10.0 (0.0)	3.0 (0.0)	2.0 (0.3)	6.9 (0.7)	10.2 (1.8)	4.7 (2.7)
Quill	5	5	11.6 (1.1)	23.8 (1.5)	6.0 (0.0)	9.6 (0.2)	458.9 (61.0)	10.6 (0.2)	8.0 (0.9)	11.2 (1.2)	11.9 (0.6)	13.1 (2.7)	9.2 (0.8)
Sugar	5	5	2.7 (0.2)	8.4 (0.7)	4.0 (0.0)	4.0 (0.0)	103.5 (39.5)	10.0 (0.0)	3.6 (0.4)	2.8 (0.5)	8.2 (0.7)	10.1 (1.9)	5.0 (1.1)
Book	5	5	301.7 (98.1)	36.4 (1.9)	6.0 (0.0)	6.8 (0.2)	5.3 (0.1)	10.2 (0.2)	5.0 (0.7)	27.2 (1.9)	21.7 (1.1)	18.8 (2.2)	16.1 (0.6)
Map	5	5	754.1 (158.2)	44.6 (2.6)	6.0 (0.0)	7.0 (0.0)	5.5 (0.2)	10.2 (0.2)	5.2 (0.4)	35.2 (2.3)	25.6 (0.5)	20.4 (2.9)	18.7 (0.6)
MilkBucket	5	5	2.8 (0.1)	6.6 (0.2)	4.0 (0.0)	3.0 (0.0)	6.9 (0.4)	10.0 (0.0)	2.0 (0.0)	2.6 (0.2)	12.5 (0.8)	13.1 (3.7)	7.4 (2.2)
BookQuill	5	5	19.8 (2.9)	19.6 (1.6)	4.0 (0.0)	4.0 (0.0)	4.3 (0.1)	10.0 (0.0)	0.8 (0.4)	15.8 (1.2)	28.4 (1.1)	2.7 (1.3)	13.5 (0.9)
MilkB.Sugar	5	5	8.8 (0.9)	12.6 (1.0)	4.0 (0.0)	4.0 (0.0)	4.0 (0.1)	10.0 (0.0)	1.2 (0.5)	8.4 (0.7)	19.3 (1.3)	3.7 (2.0)	10.7 (2.0)
Cake	5	5	344.0 (87.7)	46.2 (4.9)	5.0 (0.0)	4.8 (0.2)	2.8 (0.1)	10.0 (0.0)	2.8 (0.7)	40.4 (4.5)	44.5 (2.3)	21.8 (2.2)	17.3 (1.0)
Completed Runs		5											
Total Time (s.)		1476.8 (175.3)											
Total Calls		268.0 (6.5)											

Table 4: Results of LHRM in CraftWorld with a restricted set of callable RMs.

(a) OP

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples		Example Length	
							($\times 10^2$)	G	I	G	I
Batter	5	5	11.2 (1.6)	17.8 (1.9)	5.0 (0.0)	5.2 (0.2)	1.8 (0.1)	12.2 (0.7)	11.6 (1.4)	26.5 (2.1)	24.2 (3.2)
Bucket	5	5	0.9 (0.0)	3.6 (0.2)	3.0 (0.0)	2.0 (0.0)	1.7 (0.1)	10.0 (0.0)	1.6 (0.2)	19.4 (1.1)	19.3 (5.7)
Compass	5	5	15.5 (4.2)	18.6 (1.6)	5.0 (0.0)	5.2 (0.2)	1.8 (0.2)	11.8 (0.6)	12.8 (1.4)	28.7 (1.9)	20.3 (2.8)
Leather	5	5	0.9 (0.0)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	1.8 (0.1)	10.0 (0.0)	1.8 (0.2)	16.7 (1.7)	17.9 (4.4)
Paper	5	5	0.9 (0.0)	3.4 (0.2)	3.0 (0.0)	2.0 (0.0)	1.6 (0.1)	10.0 (0.0)	1.4 (0.2)	19.8 (2.0)	40.6 (27.0)
Quill	5	5	18.2 (3.5)	19.8 (1.2)	5.0 (0.0)	5.2 (0.2)	2.1 (0.1)	13.2 (0.4)	12.6 (1.1)	29.6 (2.5)	24.4 (3.2)
Sugar	5	5	0.8 (0.0)	3.2 (0.2)	3.0 (0.0)	2.0 (0.0)	1.7 (0.2)	10.0 (0.0)	1.2 (0.2)	17.7 (1.6)	17.5 (3.2)
Book	5	5	45.8 (4.5)	19.6 (0.9)	5.0 (0.0)	5.6 (0.2)	6.0 (0.2)	11.2 (1.0)	14.4 (0.9)	21.6 (1.8)	21.0 (1.7)
Map	5	5	64.1 (10.6)	22.0 (2.6)	5.0 (0.0)	5.2 (0.2)	6.1 (0.2)	10.8 (0.4)	17.2 (2.7)	22.5 (1.6)	23.0 (1.2)
MilkBucket	5	5	1.2 (0.1)	4.4 (0.4)	3.0 (0.0)	2.0 (0.0)	6.8 (0.3)	10.0 (0.0)	2.4 (0.4)	12.1 (0.7)	15.3 (1.6)
BookQuill	5	5	4.5 (0.8)	10.2 (1.4)	4.0 (0.0)	4.0 (0.0)	3.9 (0.1)	10.0 (0.0)	7.2 (1.4)	26.1 (0.8)	22.4 (0.9)
MilkB.Sugar	5	5	3.5 (0.5)	9.6 (1.3)	4.0 (0.0)	4.0 (0.0)	3.9 (0.1)	10.2 (0.2)	6.4 (1.2)	17.4 (0.5)	12.5 (0.8)
Cake	5	5	9.1 (0.9)	17.0 (0.9)	4.0 (0.0)	3.2 (0.2)	2.1 (0.1)	10.0 (0.0)	14.0 (0.9)	37.5 (1.9)	18.0 (1.9)
Completed Runs	5										
Total Time (s.)	176.6 (13.1)										
Total Calls	153.0 (3.6)										

(b) OPL

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							($\times 10^2$)	G	D	I	G	D	I
Batter	5	5	13.9 (3.0)	23.0 (3.0)	6.0 (0.0)	9.2 (0.2)	12.0 (1.0)	11.4 (0.4)	7.0 (1.2)	10.6 (1.6)	20.4 (1.1)	18.7 (1.6)	12.1 (1.7)
Bucket	5	5	1.8 (0.1)	7.2 (0.6)	4.0 (0.0)	4.0 (0.0)	8.0 (0.5)	10.2 (0.2)	2.2 (0.2)	2.8 (0.4)	10.2 (0.5)	13.4 (1.9)	6.8 (1.7)
Compass	5	5	13.2 (1.7)	22.0 (1.7)	6.0 (0.0)	9.2 (0.2)	10.4 (1.4)	11.0 (0.6)	6.8 (1.0)	10.2 (1.0)	17.2 (1.6)	20.9 (1.9)	14.3 (0.8)
Leather	5	5	1.9 (0.1)	7.0 (0.5)	4.0 (0.0)	4.0 (0.0)	6.9 (0.5)	10.0 (0.0)	2.4 (0.2)	2.6 (0.4)	11.1 (0.9)	16.9 (5.6)	8.9 (3.3)
Paper	5	5	2.0 (0.2)	7.6 (0.6)	4.0 (0.0)	4.0 (0.0)	7.7 (1.1)	10.0 (0.0)	3.0 (0.3)	2.6 (0.4)	10.1 (0.9)	18.9 (3.3)	5.6 (0.8)
Quill	5	5	11.5 (1.3)	22.0 (1.2)	6.0 (0.0)	9.2 (0.2)	12.8 (1.5)	10.6 (0.2)	6.4 (0.7)	11.0 (0.9)	15.3 (1.3)	13.5 (1.0)	12.1 (1.4)
Sugar	5	5	1.6 (0.1)	6.4 (0.4)	4.0 (0.0)	4.0 (0.0)	6.5 (0.7)	10.0 (0.0)	2.4 (0.2)	2.0 (0.3)	9.6 (0.6)	15.3 (3.6)	16.6 (9.2)
Book	5	5	69.0 (20.5)	21.8 (2.2)	6.0 (0.0)	6.2 (0.2)	5.5 (0.1)	10.4 (0.2)	5.2 (0.9)	12.2 (1.9)	20.4 (1.3)	21.2 (2.0)	20.8 (1.7)
Map	5	5	76.5 (6.0)	24.2 (1.3)	6.0 (0.0)	6.4 (0.2)	5.7 (0.3)	11.6 (0.8)	4.0 (0.3)	14.6 (1.0)	24.8 (3.0)	21.4 (2.1)	25.7 (0.8)
MilkBucket	5	5	1.7 (0.2)	6.0 (0.6)	4.0 (0.0)	3.0 (0.0)	7.5 (0.7)	10.2 (0.2)	1.4 (0.2)	2.4 (0.2)	11.7 (0.7)	25.4 (6.4)	14.2 (3.4)
BookQuill	5	5	5.3 (0.9)	10.8 (1.4)	4.0 (0.0)	4.0 (0.0)	3.7 (0.1)	10.0 (0.0)	1.0 (0.5)	6.8 (0.9)	27.7 (1.0)	11.2 (5.4)	21.1 (1.7)
MilkB.Sugar	5	5	4.0 (0.9)	9.8 (1.7)	4.0 (0.0)	4.0 (0.0)	3.8 (0.1)	10.0 (0.0)	1.6 (0.7)	5.2 (1.2)	18.4 (0.7)	8.3 (2.9)	15.6 (1.7)
Cake	5	5	16.2 (0.4)	20.8 (0.2)	5.0 (0.0)	4.0 (0.0)	2.1 (0.1)	10.0 (0.0)	3.2 (0.2)	14.6 (0.2)	38.1 (0.9)	22.5 (3.3)	25.8 (1.7)
Completed Runs	5												
Total Time (s.)	218.6 (21.1)												
Total Calls	188.6 (5.4)												

(c) FR

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples		Example Length	
							($\times 10^2$)	G	I	G	I
Batter	5	5	12.6 (1.8)	17.6 (1.3)	5.0 (0.0)	5.4 (0.2)	9.2 (1.2)	11.6 (0.4)	12.0 (1.2)	30.3 (2.3)	27.8 (2.0)
Bucket	5	5	1.2 (0.1)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	6.7 (0.9)	10.0 (0.0)	1.8 (0.2)	16.6 (2.5)	28.5 (4.1)
Compass	5	5	14.1 (1.5)	20.2 (1.7)	5.0 (0.0)	5.2 (0.2)	9.8 (0.7)	11.6 (0.6)	14.6 (1.2)	26.5 (0.8)	26.5 (2.1)
Leather	5	5	1.1 (0.1)	3.6 (0.2)	3.0 (0.0)	2.0 (0.0)	4.5 (0.7)	10.0 (0.0)	1.6 (0.2)	13.4 (1.3)	16.7 (3.6)
Paper	5	5	1.2 (0.1)	4.0 (0.0)	3.0 (0.0)	2.0 (0.0)	4.9 (0.9)	10.0 (0.0)	2.0 (0.0)	12.4 (1.1)	10.9 (2.5)
Quill	5	5	9.3 (0.8)	16.0 (0.8)	5.0 (0.0)	5.2 (0.2)	9.4 (1.7)	10.6 (0.2)	11.4 (0.6)	25.4 (0.3)	25.5 (2.7)
Sugar	5	5	1.4 (0.2)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	5.2 (0.3)	10.0 (0.0)	1.8 (0.2)	15.3 (1.7)	21.0 (10.1)
Book	5	5	43.8 (13.0)	20.0 (1.9)	5.0 (0.0)	5.4 (0.2)	6.0 (0.1)	10.0 (0.0)	16.0 (1.9)	21.9 (1.0)	14.7 (1.4)
Map	5	5	85.2 (13.4)	22.2 (2.5)	5.0 (0.0)	5.2 (0.2)	5.9 (0.1)	10.2 (0.2)	18.0 (2.6)	26.5 (0.9)	18.2 (1.2)
MilkBucket	5	5	1.4 (0.1)	4.4 (0.2)	3.0 (0.0)	2.0 (0.0)	10.2 (0.9)	10.0 (0.0)	2.4 (0.2)	13.0 (0.8)	12.2 (2.8)
BookQuill	5	5	6.3 (0.9)	13.2 (1.7)	4.0 (0.0)	4.0 (0.0)	3.8 (0.1)	10.0 (0.0)	10.2 (1.7)	30.6 (2.0)	11.9 (1.2)
MilkB.Sugar	5	5	4.8 (0.6)	11.8 (1.3)	4.0 (0.0)	4.0 (0.0)	3.8 (0.1)	10.0 (0.0)	8.8 (1.3)	19.8 (0.7)	8.6 (1.0)
Cake	5	5	12.5 (1.8)	20.8 (2.5)	4.0 (0.0)	3.0 (0.0)	2.3 (0.1)	10.0 (0.0)	17.8 (2.5)	44.2 (2.6)	13.1 (1.2)
Completed Runs	5										
Total Time (s.)	194.9 (17.6)										
Total Calls	161.4 (7.0)										

(d) FRL

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							($\times 10^2$)	G	D	I	G	D	I
Batter	5	5	11.2 (1.4)	23.4 (2.5)	6.0 (0.0)	9.2 (0.2)	468.4 (121.9)	10.4 (0.2)	7.6 (0.9)	11.4 (1.9)	11.9 (0.6)	10.1 (1.3)	9.9 (0.4)
Bucket	5	5	2.4 (0.1)	7.0 (0.3)	4.0 (0.0)	4.0 (0.0)	129.5 (69.4)	10.2 (0.2)	2.8 (0.2)	2.0 (0.3)	7.8 (0.5)	9.9 (1.7)	6.4 (2.1)
Compass	5	5	13.1 (1.9)	24.6 (2.2)	6.0 (0.0)	9.4 (0.2)	550.8 (156.4)	10.4 (0.2)	7.8 (1.0)	12.4 (1.2)	12.5 (1.6)	9.4 (1.0)	8.4 (0.5)
Leather	5	5	2.5 (0.4)	7.8 (0.7)	4.0 (0.0)	4.0 (0.0)	89.0 (18.0)	10.0 (0.0)	3.2 (0.4)	2.6 (0.4)	7.3 (0.4)	9.3 (1.7)	3.7 (0.4)
Paper	5	5	2.1 (0.1)	7.0 (0.3)	4.0 (0.0)	4.0 (0.0)	82.7 (18.8)	10.0 (0.0)	3.0 (0.0)	2.0 (0.3)	6.9 (0.7)	10.2 (1.8)	4.7 (2.7)
Quill	5	5	11.6 (1.2)	23.8 (1.5)	6.0 (0.0)	9.6 (0.2)	458.9 (61.0)	10.6 (0.2)	8.0 (0.9)	11.2 (1.2)	11.9 (0.6)	13.1 (2.7)	9.2 (0.8)
Sugar	5	5	2.6 (0.2)	8.4 (0.7)	4.0 (0.0)	4.0 (0.0)	103.5 (39.5)	10.0 (0.0)	3.6 (0.4)	2.8 (0.5)	8.2 (0.7)	10.1 (1.9)	5.0 (1.1)
Book	5	5	62.2 (13.2)	27.4 (2.2)	6.0 (0.0)	6.6 (0.2)	5.3 (0.1)	10.2 (0.2)	5.6 (0.6)	17.6 (1.7)	23.0 (1.0)	16.5 (2.0)	13.4 (1.0)
Map	5	5	131.3 (28.0)	34.0 (3.0)	6.0 (0.0)	6.6 (0.2)	5.5 (0.2)	10.2 (0.2)	6.8 (0.7)	23.0 (2.4)	26.2 (0.7)	16.9 (1.6)	14.5 (0.5)
MilkBucket	5	5	2.7 (0.7)	6.6 (0.6)	4.0 (0.0)	3.0 (0.0)	6.8 (0.3)	10.0 (0.0)	2.2 (0.2)	2.4 (0.4)	12.0 (0.8)	9.4 (1.0)	9.9 (2.3)
BookQuill	5	5	6.8 (0.6)	12.6 (0.7)	4.0 (0.0)	4.0 (0.0)	4.4 (0.2)	10.0 (0.0)	1.6 (0.5)	8.0 (0.3)	32.3 (2.3)	4.9 (1.5)	11.6 (1.1)
MilkB.Sugar	5	5	5.4 (0.6)	12.2 (1.0)	4.0 (0.0)	4.0 (0.0)	4.0 (0.1)	10.2 (0.2)	1.0 (0.4)	8.0 (0.4)	20.4 (1.7)	2.9 (1.4)	10.3 (1.2)
Cake	5	5	16.3 (1.2)	21.2 (1.0)	5.0 (0.0)	4.0 (0.0)	2.8 (0.0)	10.0 (0.0)	2.6 (0.2)	15.6 (1.2)	47.7 (1.9)	15.0 (0.7)	16.0 (0.9)
Completed Runs	5												
Total Time (s.)	270.1 (34.6)												
Total Calls	216.0 (5.1)												

Table 5: Results of LHRM in CRAFTWORLD without exploration using options.

(a) OP

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples		Example Length	
							($\times 10^2$)	G	I	G	I
BATTER	5	5	11.2 (1.7)	17.8 (1.9)	5.0 (0.0)	5.2 (0.2)	1.8 (0.1)	12.2 (0.7)	11.6 (1.4)	26.5 (2.1)	24.2 (3.2)
BUCKET	5	5	0.9 (0.0)	3.6 (0.2)	3.0 (0.0)	2.0 (0.0)	1.7 (0.1)	10.0 (0.0)	1.6 (0.2)	19.4 (1.1)	19.3 (5.7)
COMPASS	5	5	15.6 (4.1)	18.6 (1.6)	5.0 (0.0)	5.2 (0.2)	1.8 (0.2)	11.8 (0.6)	12.8 (1.4)	28.7 (1.9)	20.3 (2.8)
LEATHER	5	5	0.9 (0.1)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	1.8 (0.1)	10.0 (0.0)	1.8 (0.2)	16.7 (1.7)	17.9 (4.4)
PAPER	5	5	0.9 (0.1)	3.4 (0.2)	3.0 (0.0)	2.0 (0.0)	1.6 (0.1)	10.0 (0.0)	1.4 (0.2)	19.8 (2.0)	40.6 (27.0)
QUILL	5	5	18.3 (3.6)	19.8 (1.2)	5.0 (0.0)	5.2 (0.2)	2.1 (0.1)	13.2 (0.4)	12.6 (1.1)	29.6 (2.5)	24.4 (3.2)
SUGAR	5	5	0.9 (0.0)	3.2 (0.2)	3.0 (0.0)	2.0 (0.0)	1.7 (0.2)	10.0 (0.0)	1.2 (0.2)	17.7 (1.6)	17.5 (3.2)
BOOK	5	5	529.0 (164.2)	21.2 (1.4)	5.0 (0.0)	5.8 (0.2)	6.8 (0.2)	10.2 (0.2)	17.0 (1.5)	33.0 (2.6)	23.7 (1.3)
MAP	5	5	1924.2 (443.5)	28.0 (3.8)	5.0 (0.0)	5.4 (0.2)	7.8 (0.4)	10.4 (0.2)	23.6 (3.7)	40.1 (1.0)	29.4 (1.3)
MILKBUCKET	5	5	1.6 (0.2)	4.4 (0.4)	3.0 (0.0)	2.0 (0.0)	6.1 (0.3)	10.0 (0.0)	2.4 (0.4)	16.0 (1.0)	14.2 (1.3)
BOOKQUILL	5	5	42.7 (10.1)	24.6 (3.9)	4.0 (0.0)	4.0 (0.0)	6.8 (0.2)	10.0 (0.0)	21.6 (3.9)	55.8 (2.7)	21.2 (1.1)
MILKB.SUGAR	5	5	8.1 (0.8)	11.8 (1.0)	4.0 (0.0)	4.0 (0.0)	4.9 (0.1)	10.2 (0.2)	8.6 (1.2)	31.1 (0.7)	13.1 (0.8)
CAKE	5	5	198.3 (47.5)	43.0 (5.3)	4.0 (0.0)	3.8 (0.2)	5.5 (0.2)	10.0 (0.0)	40.0 (5.3)	65.0 (0.9)	22.0 (0.9)
Completed Runs	5										
Total Time (s.)	2752.8 (503.2)										
Total Calls	203.2 (11.8)										

(b) OPL

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							($\times 10^2$)	G	D	I	G	D	I
BATTER	5	5	14.1 (3.2)	23.0 (3.0)	6.0 (0.0)	9.2 (0.2)	12.0 (1.0)	11.4 (0.4)	7.0 (1.2)	10.6 (1.6)	20.4 (1.1)	18.7 (1.6)	12.1 (1.7)
BUCKET	5	5	1.8 (0.1)	7.2 (0.6)	4.0 (0.0)	4.0 (0.0)	8.0 (0.5)	10.2 (0.2)	2.2 (0.2)	2.8 (0.4)	10.2 (0.5)	13.4 (1.9)	6.8 (1.7)
COMPASS	5	5	13.5 (1.8)	22.0 (1.7)	6.0 (0.0)	9.2 (0.2)	10.4 (1.4)	11.0 (0.6)	6.8 (1.0)	10.2 (1.0)	17.2 (1.6)	20.9 (1.9)	14.3 (0.8)
LEATHER	5	5	1.8 (0.1)	7.0 (0.5)	4.0 (0.0)	4.0 (0.0)	6.9 (0.5)	10.0 (0.0)	2.4 (0.2)	2.6 (0.4)	11.1 (0.9)	16.9 (5.6)	8.9 (3.3)
PAPER	5	5	2.0 (0.2)	7.6 (0.6)	4.0 (0.0)	4.0 (0.0)	7.7 (1.1)	10.0 (0.0)	3.0 (0.3)	2.6 (0.4)	10.1 (0.9)	18.9 (3.3)	5.6 (0.8)
QUILL	5	5	11.8 (1.3)	22.0 (1.2)	6.0 (0.0)	9.2 (0.2)	12.8 (1.5)	10.6 (0.2)	6.4 (0.7)	11.0 (0.9)	15.3 (1.3)	13.5 (1.0)	12.1 (1.4)
SUGAR	5	5	1.6 (0.1)	6.4 (0.4)	4.0 (0.0)	4.0 (0.0)	6.5 (0.7)	10.0 (0.0)	2.4 (0.2)	2.0 (0.3)	9.6 (0.6)	15.3 (3.6)	16.6 (9.2)
BOOK	5	5	224.8 (71.6)	27.0 (1.9)	6.0 (0.0)	6.4 (0.2)	139.7 (21.8)	11.6 (0.4)	3.2 (0.4)	18.2 (1.4)	22.0 (1.6)	24.7 (6.5)	23.5 (1.2)
MAP	5	5	339.9 (33.6)	33.0 (2.8)	6.0 (0.0)	6.4 (0.2)	204.8 (27.1)	10.6 (0.2)	2.8 (0.5)	25.6 (2.5)	25.4 (0.8)	21.8 (3.1)	25.2 (1.1)
MILKBUCKET	5	5	3.5 (0.3)	8.2 (0.6)	4.0 (0.0)	3.0 (0.0)	47.6 (3.7)	10.2 (0.2)	2.6 (0.4)	3.4 (0.4)	10.3 (0.7)	16.2 (1.7)	14.2 (1.8)
BOOKQUILL	5	5	19.0 (2.2)	15.4 (1.5)	4.0 (0.0)	4.0 (0.0)	383.4 (83.7)	10.0 (0.0)	1.0 (0.3)	11.4 (1.3)	38.2 (1.6)	14.1 (4.9)	23.8 (1.0)
MILKB.SUGAR	5	5	11.4 (2.1)	14.4 (1.7)	4.0 (0.0)	4.0 (0.0)	87.4 (8.9)	10.4 (0.2)	1.0 (0.4)	10.0 (1.3)	19.7 (1.2)	8.7 (4.9)	17.6 (1.2)
CAKE	4	1	277.4 (0.0)	33.0 (0.0)	5.0 (0.0)	4.0 (0.0)	264.1 (0.0)	10.0 (0.0)	2.0 (0.0)	28.0 (0.0)	46.7 (0.0)	36.0 (0.0)	22.9 (0.0)
Completed Runs	5												
Total Time (s.)	701.0 (111.2)												
Total Calls	199.8 (6.9)												

(c) FRL

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							($\times 10^2$)	G	D	I	G	D	I
BATTER	5	5	11.1 (1.4)	23.4 (2.5)	6.0 (0.0)	9.2 (0.2)	468.4 (121.9)	10.4 (0.2)	7.6 (0.9)	11.4 (1.9)	11.9 (0.6)	10.1 (1.3)	9.9 (0.4)
BUCKET	5	5	2.2 (0.1)	7.0 (0.3)	4.0 (0.0)	4.0 (0.0)	129.5 (69.4)	10.2 (0.2)	2.8 (0.2)	2.0 (0.3)	7.8 (0.5)	9.9 (1.7)	6.4 (2.1)
COMPASS	5	5	12.9 (1.9)	24.6 (2.2)	6.0 (0.0)	9.4 (0.2)	550.8 (156.4)	10.4 (0.2)	7.8 (1.0)	12.4 (1.2)	12.5 (1.6)	9.4 (1.0)	8.4 (0.5)
LEATHER	5	5	2.8 (0.4)	7.8 (0.7)	4.0 (0.0)	4.0 (0.0)	89.0 (18.0)	10.0 (0.0)	3.2 (0.4)	2.6 (0.4)	7.3 (0.4)	9.3 (1.7)	3.7 (0.4)
PAPER	5	5	2.1 (0.1)	7.0 (0.3)	4.0 (0.0)	4.0 (0.0)	82.7 (18.8)	10.0 (0.0)	3.0 (0.0)	2.0 (0.3)	6.9 (0.7)	10.2 (1.8)	4.7 (2.7)
QUILL	5	5	11.6 (1.1)	23.8 (1.5)	6.0 (0.0)	9.6 (0.2)	458.9 (61.0)	10.6 (0.2)	8.0 (0.9)	11.2 (1.2)	11.9 (0.6)	13.1 (2.7)	9.2 (0.8)
SUGAR	5	5	2.6 (0.3)	8.4 (0.7)	4.0 (0.0)	4.0 (0.0)	103.5 (39.5)	10.0 (0.0)	3.6 (0.4)	2.8 (0.5)	8.2 (0.7)	10.1 (1.9)	5.0 (1.1)
BOOK	5	0	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
MAP	3	0	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
MILKBUCKET	5	2	4.7 (0.5)	11.0 (1.0)	4.0 (0.0)	3.0 (0.0)	885.6 (142.3)	10.0 (0.0)	2.0 (0.0)	7.0 (1.0)	8.2 (0.4)	10.2 (1.7)	8.8 (0.2)
BOOKQUILL	0	0	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
MILKB.SUGAR	0	0	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
CAKE	0	0	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
Completed Runs	5												
Total Time (s.)	47.1 (0.9)												
Total Calls	106.4 (2.9)												

Table 6: Results of LHRM in WATERWORLD for the default case.

(a) WOD

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples		Example Length	
							$(\times 10^2)$	G	I	G	I
RG	5	5	0.9 (0.0)	4.0 (0.0)	3.0 (0.0)	2.0 (0.0)	0.9 (0.1)	10.0 (0.0)	2.0 (0.0)	11.2 (1.0)	5.8 (1.1)
BC	5	5	0.9 (0.1)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	0.8 (0.1)	10.0 (0.0)	1.8 (0.2)	10.8 (0.8)	11.9 (3.4)
MY	5	5	0.9 (0.0)	3.6 (0.2)	3.0 (0.0)	2.0 (0.0)	0.7 (0.0)	10.0 (0.0)	1.6 (0.2)	8.7 (0.8)	6.6 (1.9)
RG&BC	5	5	4.5 (0.3)	13.4 (0.4)	4.0 (0.0)	4.0 (0.0)	8.8 (0.3)	11.8 (0.6)	8.6 (0.7)	12.2 (0.9)	14.8 (1.2)
BC&MY	5	5	5.8 (1.0)	15.6 (2.1)	4.0 (0.0)	4.0 (0.0)	8.1 (0.2)	12.8 (1.3)	9.8 (1.5)	13.2 (1.7)	17.1 (1.6)
RG&MY	5	5	4.7 (0.5)	13.2 (1.0)	4.0 (0.0)	4.0 (0.0)	8.5 (0.2)	10.8 (0.2)	9.4 (0.9)	12.2 (0.7)	18.6 (1.2)
RGB	5	5	1.2 (0.1)	4.8 (0.5)	3.0 (0.0)	2.0 (0.0)	8.6 (0.2)	10.0 (0.0)	2.8 (0.5)	7.8 (0.2)	7.0 (1.4)
CMY	5	5	1.4 (0.2)	5.4 (0.7)	3.0 (0.0)	2.0 (0.0)	8.8 (0.5)	10.0 (0.0)	3.4 (0.7)	8.0 (0.3)	10.2 (1.3)
RGB&CMY	5	5	15.1 (1.7)	21.6 (1.7)	4.0 (0.0)	4.0 (0.0)	2.3 (0.0)	11.0 (0.4)	17.6 (1.7)	17.3 (0.4)	22.6 (1.6)
Completed Runs	5										
Total Time (s.)	35.4 (2.0)										
Total Calls	85.4 (3.1)										

(b) WD

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							$(\times 10^2)$	G	D	I	G	D	I
RG	5	5	1.9 (0.2)	7.8 (1.1)	4.0 (0.0)	4.0 (0.0)	3.0 (0.3)	10.0 (0.0)	3.0 (0.3)	2.8 (0.9)	7.0 (0.7)	10.3 (1.6)	4.2 (0.8)
BC	5	5	1.8 (0.3)	7.2 (1.0)	4.0 (0.0)	4.0 (0.0)	2.7 (0.3)	10.2 (0.2)	2.4 (0.2)	2.6 (0.7)	8.4 (0.5)	6.9 (1.5)	6.3 (1.7)
MY	5	5	1.4 (0.1)	5.8 (0.4)	4.0 (0.0)	4.0 (0.0)	2.9 (0.3)	10.0 (0.0)	2.4 (0.2)	1.4 (0.2)	6.9 (0.4)	5.8 (1.5)	4.6 (1.6)
RG&BC	5	5	11.7 (2.5)	24.0 (3.8)	4.8 (0.2)	4.8 (0.2)	12.0 (0.4)	13.0 (0.7)	6.2 (1.6)	11.8 (1.9)	11.7 (0.8)	6.9 (1.5)	11.8 (0.8)
BC&MY	5	5	9.5 (1.5)	20.8 (2.4)	4.8 (0.2)	4.8 (0.2)	11.5 (0.4)	11.2 (0.6)	5.2 (0.7)	11.4 (1.6)	10.6 (0.9)	8.8 (1.4)	13.2 (0.9)
RG&MY	5	5	5.4 (0.5)	14.0 (1.3)	4.2 (0.2)	4.2 (0.2)	11.8 (0.6)	10.6 (0.2)	3.2 (0.7)	7.2 (1.1)	9.8 (0.3)	6.2 (1.9)	11.6 (0.7)
RGB	5	5	2.5 (0.3)	8.2 (0.7)	4.0 (0.0)	3.0 (0.0)	11.9 (0.6)	10.2 (0.2)	3.0 (0.3)	3.0 (0.5)	7.9 (0.4)	14.0 (1.8)	10.6 (2.0)
CMY	5	5	3.6 (0.4)	11.4 (1.2)	4.0 (0.0)	3.0 (0.0)	10.8 (0.3)	10.0 (0.0)	4.2 (0.7)	5.2 (0.6)	7.9 (0.2)	8.8 (1.6)	10.5 (1.1)
RGB&CMY	5	5	29.0 (4.1)	31.4 (2.8)	4.4 (0.2)	4.4 (0.2)	4.9 (0.3)	11.2 (0.4)	5.4 (1.6)	21.8 (1.3)	16.6 (0.8)	7.4 (0.9)	17.2 (0.6)
Completed Runs	5												
Total Time (s.)	67.0 (6.2)												
Total Calls	130.6 (6.0)												

Table 7: Results of LHRM in WATERWORLD with a restricted set of callable RMs.

(a) WOD

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples		Example Length	
							$(\times 10^2)$	G	I	G	I
RG	5	5	0.9 (0.0)	4.0 (0.0)	3.0 (0.0)	2.0 (0.0)	0.9 (0.1)	10.0 (0.0)	2.0 (0.0)	11.2 (1.0)	5.8 (1.1)
BC	5	5	0.9 (0.1)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	0.8 (0.1)	10.0 (0.0)	1.8 (0.2)	10.8 (0.8)	11.9 (3.4)
MY	5	5	0.9 (0.0)	3.6 (0.2)	3.0 (0.0)	2.0 (0.0)	0.7 (0.0)	10.0 (0.0)	1.6 (0.2)	8.7 (0.8)	6.6 (1.9)
RG&BC	5	5	5.3 (0.4)	15.2 (0.9)	4.0 (0.0)	4.0 (0.0)	8.6 (0.3)	12.4 (0.2)	9.8 (0.8)	14.7 (1.3)	16.0 (0.8)
BC&MY	5	5	3.9 (0.1)	12.4 (0.2)	4.0 (0.0)	4.0 (0.0)	8.3 (0.4)	11.8 (0.7)	7.6 (0.7)	11.2 (0.8)	13.2 (1.0)
RG&MY	5	5	4.6 (0.3)	13.8 (0.9)	4.0 (0.0)	4.0 (0.0)	8.5 (0.2)	10.2 (0.2)	10.6 (0.9)	10.7 (0.5)	15.8 (1.6)
RGB	5	5	1.2 (0.1)	4.8 (0.7)	3.0 (0.0)	2.0 (0.0)	8.7 (0.2)	10.2 (0.2)	2.6 (0.7)	8.3 (0.5)	16.2 (3.8)
CMY	5	5	1.6 (0.2)	6.2 (0.7)	3.0 (0.0)	2.0 (0.0)	8.6 (0.5)	10.0 (0.0)	4.2 (0.7)	8.0 (0.3)	10.8 (1.2)
RGB&CMY	5	5	5.7 (0.8)	15.0 (1.6)	4.0 (0.0)	4.0 (0.0)	2.6 (0.1)	10.4 (0.4)	11.6 (1.6)	17.0 (1.1)	15.9 (1.3)
Completed Runs	5										
Total Time (s.)	24.9 (0.9)										
Total Calls	78.8 (2.7)										

(b) WD

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							$(\times 10^2)$	G	D	I	G	D	I
RG	5	5	1.9 (0.3)	7.8 (1.1)	4.0 (0.0)	4.0 (0.0)	3.0 (0.3)	10.0 (0.0)	3.0 (0.3)	2.8 (0.9)	7.0 (0.7)	10.3 (1.6)	4.2 (0.8)
BC	5	5	1.8 (0.3)	7.2 (1.0)	4.0 (0.0)	4.0 (0.0)	2.7 (0.3)	10.2 (0.2)	2.4 (0.2)	2.6 (0.7)	8.4 (0.5)	6.9 (1.5)	6.3 (1.7)
MY	5	5	1.4 (0.1)	5.8 (0.4)	4.0 (0.0)	4.0 (0.0)	2.9 (0.3)	10.0 (0.0)	2.4 (0.2)	1.4 (0.2)	6.9 (0.4)	5.8 (1.5)	4.6 (1.6)
RG&BC	5	5	6.9 (0.7)	17.6 (1.5)	4.6 (0.2)	4.6 (0.2)	12.0 (0.4)	10.8 (0.2)	4.6 (0.5)	9.2 (1.0)	10.4 (0.5)	9.4 (1.8)	12.6 (0.7)
BC&MY	5	5	9.3 (1.8)	21.4 (2.9)	4.8 (0.2)	4.8 (0.2)	11.7 (0.5)	12.2 (1.0)	5.8 (1.1)	10.4 (1.8)	11.4 (0.7)	6.9 (1.3)	12.1 (0.7)
RG&MY	5	5	7.8 (1.1)	18.8 (1.9)	4.8 (0.2)	4.8 (0.2)	11.8 (0.5)	11.0 (0.3)	4.8 (0.4)	10.0 (2.0)	9.8 (0.2)	8.6 (0.8)	13.0 (0.8)
RGB	5	5	2.1 (0.1)	7.6 (0.2)	4.0 (0.0)	3.0 (0.0)	11.9 (0.5)	10.0 (0.0)	2.6 (0.2)	3.0 (0.0)	7.6 (0.5)	11.8 (1.7)	10.7 (1.7)
CMY	5	5	2.3 (0.2)	8.2 (0.8)	4.0 (0.0)	3.0 (0.0)	10.7 (0.2)	10.0 (0.0)	2.2 (0.4)	4.0 (0.5)	7.8 (0.2)	9.9 (1.6)	8.9 (0.5)
RGB&CMY	5	5	9.6 (1.5)	20.6 (2.6)	5.0 (0.0)	5.0 (0.0)	5.0 (0.6)	10.2 (0.2)	6.4 (0.9)	11.0 (1.8)	15.0 (0.5)	12.3 (0.8)	14.2 (1.2)
Completed Runs	5												
Total Time (s.)	42.9 (3.7)												
Total Calls	115.0 (7.5)												

Table 8: Results of LHRM in WATERWORLD without exploration using options.

(a) WOD

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples		Example Length	
							($\times 10^2$)	G	I	G	I
RG	5	5	0.9 (0.0)	4.0 (0.0)	3.0 (0.0)	2.0 (0.0)	0.9 (0.1)	10.0 (0.0)	2.0 (0.0)	11.2 (1.0)	5.8 (1.1)
BC	5	5	0.9 (0.1)	3.8 (0.2)	3.0 (0.0)	2.0 (0.0)	0.8 (0.1)	10.0 (0.0)	1.8 (0.2)	10.8 (0.8)	11.9 (3.4)
MY	5	5	0.9 (0.0)	3.6 (0.2)	3.0 (0.0)	2.0 (0.0)	0.7 (0.0)	10.0 (0.0)	1.6 (0.2)	8.7 (0.8)	6.6 (1.9)
RG&BC	5	5	4.2 (0.4)	12.2 (0.9)	4.0 (0.0)	4.0 (0.0)	9.5 (0.3)	10.6 (0.2)	8.6 (1.1)	13.8 (0.2)	15.3 (1.6)
BC&MY	5	5	4.3 (0.3)	11.8 (0.7)	4.0 (0.0)	4.0 (0.0)	9.8 (0.1)	11.6 (0.2)	7.2 (0.6)	15.3 (0.9)	16.7 (1.7)
RG&MY	5	5	4.6 (0.3)	12.6 (0.7)	4.0 (0.0)	4.0 (0.0)	9.5 (0.1)	11.2 (0.4)	8.4 (0.7)	14.2 (0.9)	14.8 (0.8)
RGB	5	5	1.2 (0.2)	4.6 (0.7)	3.0 (0.0)	2.0 (0.0)	9.0 (0.1)	10.0 (0.0)	2.6 (0.7)	9.4 (0.4)	8.7 (1.7)
CMY	5	5	1.4 (0.1)	5.0 (0.5)	3.0 (0.0)	2.0 (0.0)	8.8 (0.2)	10.0 (0.0)	3.0 (0.5)	8.8 (0.2)	10.6 (1.5)
RGB&CMY	5	5	16.1 (1.1)	19.8 (1.1)	4.0 (0.0)	4.0 (0.0)	4.1 (0.1)	11.2 (0.6)	15.6 (1.3)	26.0 (1.2)	21.6 (0.9)

Completed Runs	5
Total Time (s.)	34.4 (1.4)
Total Calls	77.4 (2.0)

(b) WD

Task	# G	# L	Time (s.)	Calls	States	Edges	Ep. First HRM	# Examples			Example Length		
							($\times 10^2$)	G	D	I	G	D	I
RG	5	5	1.9 (0.3)	7.8 (1.1)	4.0 (0.0)	4.0 (0.0)	3.0 (0.3)	10.0 (0.0)	3.0 (0.3)	2.8 (0.9)	7.0 (0.7)	10.3 (1.6)	4.2 (0.8)
BC	5	5	1.8 (0.2)	7.2 (1.0)	4.0 (0.0)	4.0 (0.0)	2.7 (0.3)	10.2 (0.2)	2.4 (0.2)	2.6 (0.7)	8.4 (0.5)	6.9 (1.5)	6.3 (1.7)
MY	5	5	1.4 (0.1)	5.8 (0.4)	4.0 (0.0)	4.0 (0.0)	2.9 (0.3)	10.0 (0.0)	2.4 (0.2)	1.4 (0.2)	6.9 (0.4)	5.8 (1.5)	4.6 (1.6)
RG&BC	5	5	8.1 (1.4)	18.2 (2.2)	4.6 (0.2)	4.6 (0.2)	97.4 (4.2)	10.8 (0.4)	5.2 (0.6)	9.2 (1.4)	10.5 (0.5)	10.3 (1.7)	13.7 (1.5)
BC&MY	5	5	6.2 (0.5)	15.6 (0.7)	4.6 (0.2)	4.6 (0.2)	91.5 (5.8)	10.6 (0.2)	4.6 (0.6)	7.4 (0.7)	9.7 (0.2)	7.0 (1.2)	11.3 (1.0)
RG&MY	5	5	8.6 (1.8)	19.2 (2.7)	4.4 (0.2)	4.4 (0.2)	90.3 (5.3)	11.2 (0.8)	5.6 (0.9)	9.4 (1.3)	10.4 (0.7)	7.7 (0.7)	13.3 (0.9)
RGB	5	5	2.3 (0.1)	7.6 (0.2)	4.0 (0.0)	3.0 (0.0)	65.3 (1.6)	10.2 (0.2)	2.6 (0.4)	2.8 (0.4)	7.6 (0.3)	11.1 (3.0)	8.8 (1.7)
CMY	5	5	4.4 (0.6)	13.2 (1.5)	3.8 (0.2)	3.0 (0.0)	59.2 (2.9)	10.2 (0.2)	3.8 (0.6)	7.2 (1.0)	6.9 (0.5)	8.2 (1.0)	7.6 (0.8)
RGB&CMY	5	5	32.1 (5.0)	31.4 (3.3)	4.4 (0.2)	4.4 (0.2)	125.7 (9.9)	11.4 (0.5)	5.8 (1.2)	21.2 (2.2)	17.1 (0.6)	8.4 (1.6)	17.8 (1.0)

Completed Runs	5
Total Time (s.)	66.7 (6.6)
Total Calls	126.0 (6.3)

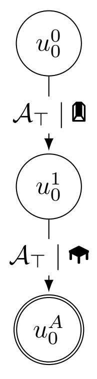
Learning Flat HRMs. Table 9 shows the results of learning a non-flat HRM using LHRM, and the results of learning a flat HRM using several approaches (LHRM, DeepSynth, LRM and JIRP). An extended discussion of these results can be found in Section 6.

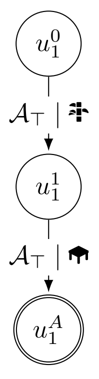
H Examples of Hierarchies of Reward Machines

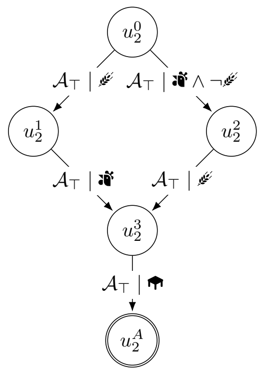
Figures 11 and 12 show the minimal root RMs for the CRAFTWORLD and WATERWORLD tasks, respectively. In the case of CRAFTWORLD, since two or more propositions can never occur simultaneously, the mutual exclusivity between formulas could be enforced differently. These RMs correspond to the settings without dead-ends; thus, they do not include rejecting states.

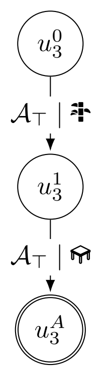
Table 9: Results of learning non-flat and flat HRMs using different methods. The columns are the following: the number of completed runs without timing out, the amount of time needed to learn the HRMs or RMs, and the number of states and edges of the RM.

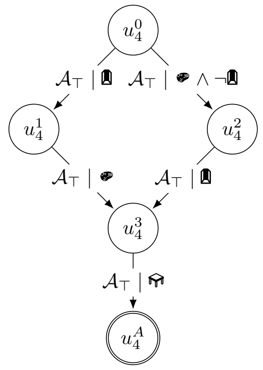
Task	LHRM (Non-Flat)				LHRM (Flat)				DeepSynth				LRM				JIRP			
	C	Time (s.)	States	Edges	C	Time (s.)	States	Edges	C	Time (s.)	States	Edges	C	Time (s.)	States	Edges	C	Time (s.)	States	Edges
MILKBUCKET	5	1.5 (0.2)	3.0 (0.0)	2.0 (0.0)	5	3.2 (0.6)	4.0 (0.0)	3.6 (0.2)	5	325.6 (29.7)	13.4 (0.4)	93.2 (1.7)	5	347.5 (64.5)	4.0 (0.0)	14.0 (1.0)	5	17.1 (5.5)	4.0 (0.0)	3.0 (0.0)
BOOK	5	191.2 (36.4)	5.0 (0.0)	5.8 (0.2)	0	-	-	-	5	288.9 (31.7)	16.6 (3.1)	119.0 (19.4)	5	2261.0 (552.2)	8.0 (0.0)	31.2 (2.0)	0	-	-	-
BOOKQUILL	5	17.9 (1.4)	4.0 (0.0)	4.0 (0.0)	0	-	-	-	5	308.6 (52.6)	12.8 (0.5)	92.8 (2.3)	0	-	-	-	0	-	-	-
CAKE	5	74.5 (25.7)	4.0 (0.0)	3.2 (0.2)	0	-	-	-	4	290.6 (36.4)	17.2 (2.5)	110.2 (11.6)	0	-	-	-	0	-	-	-
RG	5	0.9 (0.0)	3.0 (0.0)	2.0 (0.0)	5	0.9 (0.0)	3.0 (0.0)	2.0 (0.0)	0	-	-	-	0	-	-	-	5	32.3 (7.9)	3.8 (0.2)	82.4 (9.1)
RG&BC	5	4.5 (0.3)	4.0 (0.0)	4.0 (0.0)	0	-	-	-	0	-	-	-	0	-	-	-	0	-	-	-
RGB&CMY	5	15.1 (1.7)	4.0 (0.0)	4.0 (0.0)	0	-	-	-	0	-	-	-	0	-	-	-	0	-	-	-

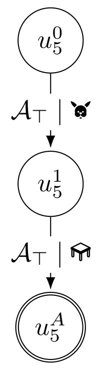

(a) $\mathcal{A}_0$ – BUCKET

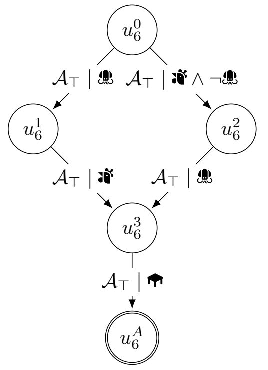

(b) $\mathcal{A}_1$ – SUGAR

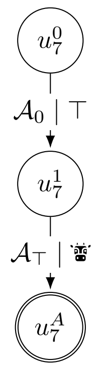

(c) $\mathcal{A}_2$ – BATTER

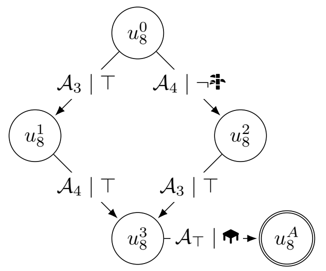

(d) $\mathcal{A}_3$ – PAPER

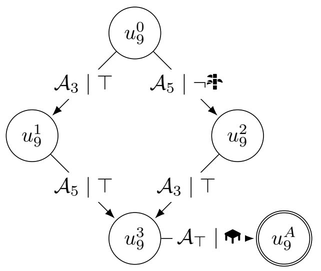

(e) $\mathcal{A}_4$ – COMPASS

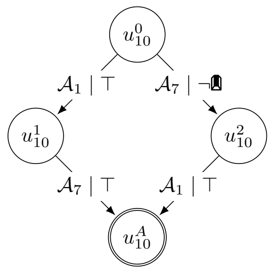

(f) $\mathcal{A}_5$ – LEATHER

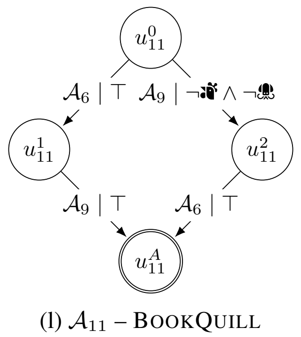

(g) $\mathcal{A}_6$ – QUILL


(h) $\mathcal{A}_7$ – MILKBUCKET


(i) $\mathcal{A}_8$ – MAP


(j) $\mathcal{A}_9$ – BOOK


(k) $\mathcal{A}_{10}$ – MILKB.SUGAR


(l) $\mathcal{A}_{11}$ – BOOKQUILL

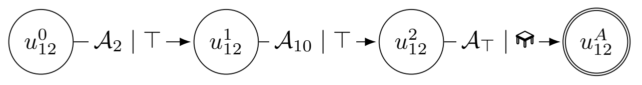
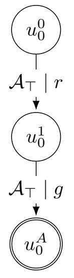
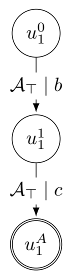

(m) $\mathcal{A}_{12}$ – CAKE

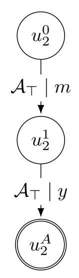
Figure 11: Root reward machines for each of the CRAFTWORLD tasks.



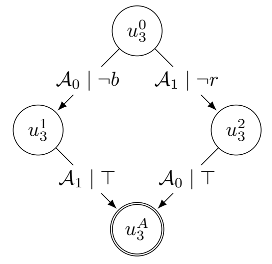
(a) $\mathcal{A}_0$ – RG



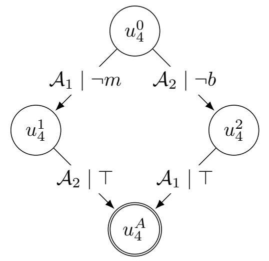
(b) $\mathcal{A}_1$ – BC



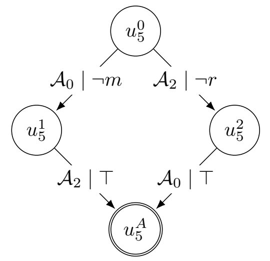
(c) $\mathcal{A}_2$ – MY


u_3^0
$\mathcal{A}_0 \mid \neg b$
$\mathcal{A}_1 \mid \neg r$
u_3^1
u_3^2
$\mathcal{A}_1 \mid \top$
$\mathcal{A}_0 \mid \top$
u_3^A

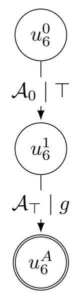
(d) $\mathcal{A}_3$ – RG&BC


u_4^0
$\mathcal{A}_1 \mid \neg m$
$\mathcal{A}_2 \mid \neg b$
u_4^1
u_4^2
$\mathcal{A}_2 \mid \top$
$\mathcal{A}_1 \mid \top$
u_4^A

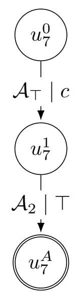
(e) $\mathcal{A}_4$ – BC&MY


u_5^0
$\mathcal{A}_0 \mid \neg m$
$\mathcal{A}_2 \mid \neg r$
u_5^1
u_5^2
$\mathcal{A}_2 \mid \top$
$\mathcal{A}_0 \mid \top$
u_5^A

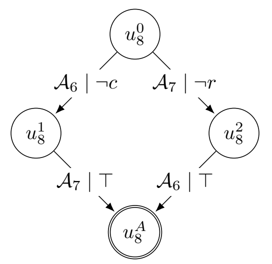
(f) $\mathcal{A}_5$ – RG&MY



(g) $\mathcal{A}_6$ – RGB



(h) $\mathcal{A}_7$ – CMY


u_8^0
$\mathcal{A}_6 \mid \neg c$
$\mathcal{A}_7 \mid \neg r$
u_8^1
u_8^2
$\mathcal{A}_7 \mid \top$
$\mathcal{A}_6 \mid \top$
u_8^A

(i) $\mathcal{A}_8$ – RGB&CMY

Figure 12: Root reward machines for each of the WATERWORLD tasks.