Hierarchies of Reward Machines

Daniel Furelos-Blanco
Department of Computing
Imperial College London
d.furelos-blanco18@imperial.ac.uk

Mark Law
ILASP Limited
mark@ilasp.com

Anders Jonsson
Department of Information and Communication Technologies
Universitat Pompeu Fabra
anders.jonsson@upf.edu

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

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

Abstract

Hierarchical reinforcement learning (HRL) algorithms decompose a task into simpler subtasks that can be independently solved. This enables tackling complex long-horizon and/or sparse reward tasks more efficiently. In recent years, several efforts have focused on proposing discrete structures, such as finite-state machines (FSMs), that can be exploited using HRL and learned from an agent's experience. In this paper, we introduce a formalism for hierarchically composing reward machines (RMs). RMs are FSMs where each edge is labeled by (1) a propositional logic formula over a set of high-level events that capture a task's landmark/subgoal, and (2) a reward for satisfying the formula. The structure of an RM is naturally exploited by HRL algorithms by treating each landmark as a subtask and deciding which subtask to pursue from each RM state. A hierarchy of reward machines (HRM) enables the constituent RMs to call each other, potentially defining an arbitrary number of increasingly abstract machines. Our formalism guarantees that an HRM can be converted into an equivalent flat one. We adapt HRL algorithms to HRMs by defining each RM in the hierarchy as a subtask itself. Given a set of tasks with hierarchical structure, we describe a curriculum-based method to induce an HRM for each task in the set. Each HRM is induced from a set of traces of high-level events and a set of callable RMs from lower level tasks. We evaluate our method in two domains with hierarchically composable tasks. We show that encapsulating each task's structure within an HRM makes the learning of a multi-level HRM more efficient than that of a flat HRM since the size of the root machine is potentially much smaller. We also study how efficient it is to use HRMs from lower levels to drive the search for example traces in higher level tasks.

Keywords: Automaton Learning
Hierarchical Reinforcement Learning
Reward Machines

Acknowledgements

The authors would like to thank the anonymous reviewers, Hadeel Al-Negheimish and Alex Spies for their helpful comments and suggestions. Anders Jonsson is partially funded by Spanish grant PID2019-108141GB-I00.

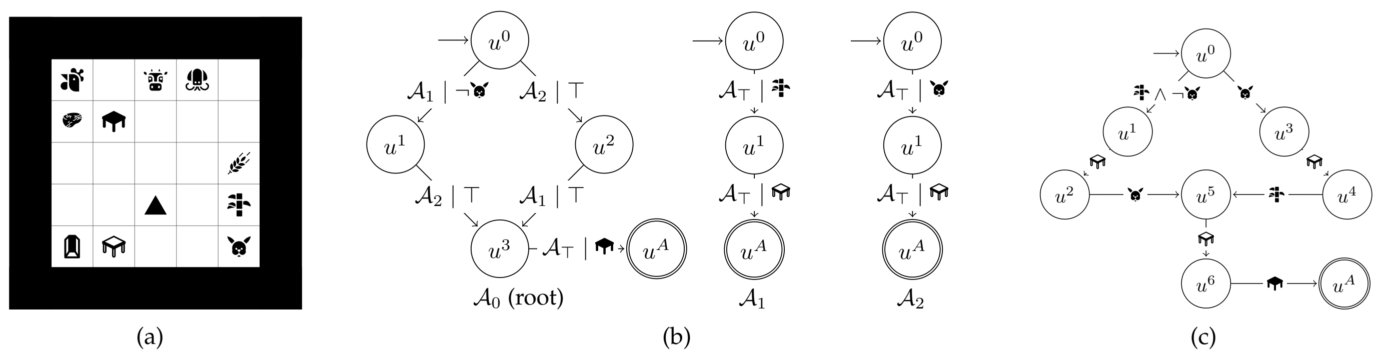
(a)
u^0
$\mathcal{A}_1 \mid \neg$🐰
$\mathcal{A}_2 \mid \top$
u^1
u^2
$\mathcal{A}_2 \mid \top$
$\mathcal{A}_1 \mid \top$
u^3
$\mathcal{A}_\top \mid$ 🪑
u^A
$\mathcal{A}_0$ (root)
$\mathcal{A}_\top \mid$ 🎋
$\mathcal{A}_\top \mid$ 🛠
$\mathcal{A}_1$
$\mathcal{A}_\top \mid$ 🐰
$\mathcal{A}_2$
(b)
🎋 $\wedge \neg$🐰
u^4
u^5
u^6
(c)

Figure 1: A CRAFTWORLD grid (a), a multi-level HRM for BOOK (b) and an equivalent flat HRM (c).

Task	Level	Description	Task	Level	Description	Task	Level	Description	Task	Level	Description
BATTER	1	(🌾 & 🐔) ; 🪑	PAPER	1	🎋 ; 🛠	MAP	2	(PAPER & COMPASS) ; 🪑	CAKE	4	BATTER ; MILKB.SUGAR ; 🛠
BUCKET	1	▯ ; 🪑	QUILL	1	(🦑 & 🐔); 🪑	MILKBUCKET	2	BUCKET ; 🐄			
COMPASS	1	(▯ & ●) ; 🛠	SUGAR	1	🎋; 🪑	BOOKQUILL	3	BOOK & QUILL			
LEATHER	1	🐰 ; 🛠	BOOK	2	(PAPER & LEATHER) ; 🪑	MILKB.SUGAR	3	MILKBUCKET & SUGAR			

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 done in any order.

1 Introduction

Reward machines (RMs) [8] are a recent formalism for tackling sparse reward tasks in partially observable environments by encoding the reward function of a given task. RMs are finite-state machines whose edges are labeled by propositional logic formulas over a set of high-level events and a reward scalar. RMs capture a task's subgoals through these formulas, and thus help to address partial observability by acting as an external memory. These structures are amenable to the use of hierarchical reinforcement learning (HRL) frameworks, such as options [7], by associating an option to each formula in the RM. Recent papers have proposed methods for learning RMs [9, 11, 4] and similar kinds of machines [3, 2]. The primary shortcoming of the RMs considered by previous work is that they cannot be reused within larger RMs, thus the same policies might be learned multiple times unnecessarily. Besides, methods for learning RMs do not usually scale well when the RMs consist of several states. In this work, we propose a formalism for hierarchically composing RMs by allowing calls between them. We introduce a curriculum-based method for inducing these hierarchies given a set of tasks classified into different levels according to their subtasks. Hierarchies of RMs (HRMs) enable reusability and ease the machine induction process since the constituent RMs are smaller. Our method successfully learns and exploits HRMs in environments with hierarchically composable tasks, outperforming the learning of equivalent flat RMs. We empirically show that using previously learned HRMs to explore allows for a more efficient collection of example traces in new tasks.

Preliminaries An episodic partially observable Markov decision process (POMDP) is a tuple $\mathcal{M} = \langle S, S_T, S_G, \Sigma, A, p, r, \gamma, \nu \rangle$, where S, A, p, r and γ are defined as for MDPs, $S_T \subseteq S$ is a set of terminal states, $S_G \subseteq S_T$ is a set of goal states, Σ is a set of observations, and $\nu : S \to \Delta(\Sigma)$ is a mapping from states to probability distributions over observations. The POMDP is enhanced with a set of *propositions* $\mathcal{P}$, and a *labeling function* $\mathcal{L} : \Sigma \to 2^{\mathcal{P}}$ mapping observations into subsets of propositions (or *labels*) $L \subseteq \mathcal{P}$. The aim is to find a policy $\pi : (\Sigma \times A)^* \times \Sigma \to A$, a mapping from histories of observation-action pairs to actions, which maximizes the expected sum of discounted rewards (or *return*), $R_t = \mathbb{E}_\pi[\sum_{k=t}^{n} \gamma^{k-t} r_t]$, where n is the last step of the episode. We assume that the combination of an observation and a history of labels seen during an episode is sufficient to obtain the Markov property, i.e. a policy can be defined as $\pi : (2^{\mathcal{P}})^* \times \Sigma \to A$.

The interaction between the agent and a POMDP environment is as follows. At time t, the state of the environment is $s_t \in S$, and the agent observes a tuple $\boldsymbol{\sigma}_t = \langle \sigma_t^\Sigma, \sigma_t^T, \sigma_t^G \rangle$, where $\sigma_t^\Sigma \sim \nu(\cdot|s_t)$ is an observation, and $\sigma_t^T = \mathbb{I}[s_t \in S_T]$ and $\sigma_t^G = \mathbb{I}[s_t \in S_G]$ indicate whether s_t is a terminal state and a goal state respectively. If the state is non-terminal, the agent executes action $a_t \in A$, the environment transitions to state $s_{t+1} \sim p(\cdot|s_t, a_t)$, and the agent observes a new tuple $\boldsymbol{\sigma}_{t+1}$ and receives reward $r(s_t, a_t, s_{t+1})$. At the end of the episode, we will have a trace $\lambda = \langle \boldsymbol{\sigma}_0, a_0, r_1, \boldsymbol{\sigma}_1, a_1, \ldots, a_{n-1}, r_n, \boldsymbol{\sigma}_n \rangle$, which can be of three types: a goal trace if $\sigma_n^G = \top$, a dead-end trace if $\sigma_n^T = \top \wedge \sigma_n^G = \bot$, and incomplete if $\sigma_n^T = \bot$. A label trace $\lambda_{\mathcal{L},\mathcal{P}}$ can be derived by applying the labeling function $\mathcal{L}$ to each observation $\sigma_{0 \le i \le n}^\Sigma$ in the previous trace.

The options framework [7] addresses temporal abstraction in reinforcement learning. An option is a tuple $\omega = \langle I_\omega, \pi_\omega, \beta_\omega \rangle$, where I_ω and β_ω respectively denote where the option initiates and terminates (e.g., a subset of states), and π_ω is the option's policy describing the behavior of the option between I_ω and β_ω.

2 Contributions

We propose the CraftWorld domain (cf. Figure 1a) to describe our method. 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 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 goal is achieved and 0 otherwise. A label L is used as a truth assignment where propositions $p \in \mathcal{P}$ in L are true, else they are false (e.g., {🎋} satisfies 🎋 $\wedge \neg$🐰).

Formalism A *hierarchy of reward machines (HRM)* is a tuple $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}, \delta_\mathcal{H} \rangle$, where $\mathcal{A} = \{\mathcal{A}_0, \ldots, \mathcal{A}_{m-1}\} \cup \{\mathcal{A}_\top\}$ is a set of m RMs and a leaf RM $\mathcal{A}_\top$, $\mathcal{A}_r \in \mathcal{A} \setminus \{\mathcal{A}_\top\}$ is the root RM, $\mathcal{P}$ is a finite set of propositions shared by all RMs, and $\delta_\mathcal{H} : U_\mathcal{H} \times 2^\mathcal{P} \to U_\mathcal{H}$ is a hierarchical transition function. The set $U_\mathcal{H}$ denotes the set of all possible *hierarchy states*, each a tuple $\langle \mathcal{A}_i, u, \Gamma \rangle$ where $\mathcal{A}_i \in \mathcal{A}$ is an RM, u is a state of $\mathcal{A}_i$, and Γ is a call stack. The call stack determines the RMs to which control must be returned once a call is completed. Each *reward machine (RM)* $\mathcal{A}_i \in \mathcal{A}$ is a tuple $\mathcal{A}_i = \langle U_i, \mathcal{P}, \varphi_i, r_i, u_i^0, U_i^A, U_i^R \rangle$, where U_i is a finite set of states, $\mathcal{P}$ is a finite set of propositions, $\varphi_i : U_i \times U_i \times \mathcal{A} \to \text{DNF}_\mathcal{P}$ is the state transition function, $r_i : U_i \times U_i \to \mathbb{R}$ is the reward transition function, $u_i^0 \in U_i$ is the initial state, $U_i^A \subseteq U_i$ is the set of accepting states, and $U_i^R \subseteq U_i$ is the set of rejecting states.[1,2] The *leaf machine* $\mathcal{A}_\top$ has a single state, which is accepting (i.e., $U_\top = U_\top^A = \{u_\top^0\}$). The expression $\varphi_i(u, u', \mathcal{A}_j) = \phi$ indicates that the transition from $u \in U_i$ to $u' \in U_i$ is associated with a call to RM $\mathcal{A}_j$ and DNF formula $\phi \in \text{DNF}_\mathcal{P}$, which must be satisfied to start the call (by default, $\phi = \bot$). Accepting and rejecting states do not have transitions to other states. Figure 1b shows Book's HRM, which consists of a root and two RMs for the subtasks Paper and Leather. 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.

The execution of an HRM starts in the root's initial state with an empty call stack. Given a hierarchy state and a label $L \subseteq \mathcal{P}$, the next hierarchy state is determined by the hierarchical transition function $\delta_\mathcal{H}$, which is recursively defined using the transition functions φ_i of the constituent RMs. Given a hierarchy state $\langle \mathcal{A}_i, u, \Gamma \rangle$, $\delta_\mathcal{H}$ covers three cases:

1. If $u \in U_i^A$ is accepting and Γ is non-empty, pop the top element of Γ and return control to the previous RM on the call stack, recursively applying $\delta_\mathcal{H}$ in case several accepting states are reached simultaneously.
2. If L satisfies the formula ϕ of a transition $\varphi_i(u, u', \mathcal{A}_j) = \phi$, as well as formulas from initial states of recursively called RMs, push $\mathcal{A}_j$ onto Γ and recursively apply $\delta_\mathcal{H}$ from its initial state. In Figure 1b, L must satisfy 🎋 $\wedge \neg$🐰 to start $\mathcal{A}_1$ from the root's initial state, while it only needs to satisfy 🎋 if the call is made from state u^2.
3. If none of the conditions in previous cases hold, the hierarchy state does not change.

The theorem below captures that the behavior of $\delta_\mathcal{H}$ is equivalent to the transition functions of flat RMs in previous works, specifically those using logic formulas to label the edges [2], so an HRM cannot have circular dependencies and must behave deterministically (two state transitions cannot be satisfied at once). We omit the proof due to space constraints.

Theorem 1. *Every HRM $\mathcal{H}$ and associated hierarchical transition function $\delta_\mathcal{H}$ corresponds to an equivalent flat RM.*

Reinforcement Learning An HRM can be exploited using options, similar to flat finite-state machines [8, 2]. Each transition $\varphi_i(u, u', \mathcal{A}_j) = \phi$ is associated with a *formula option* ϕ if $\mathcal{A}_j = \mathcal{A}_\top$, and a *call option* $\langle \mathcal{A}_j, \phi \rangle$ if $\mathcal{A}_j \neq \mathcal{A}_\top$. Both types of options are applicable in RM state u of $\mathcal{A}_i$. A formula option simply attempts to satisfy ϕ, while a call option additionally has to satisfy formulas in recursively called RMs. In Figure 1b, the set of formula options is {🐰, 🎋 $\wedge \neg$🐰, 🎋, 🪑, 🛠}, and option 🎋 $\wedge \neg$🐰 leads the agent to observe label {🎋}. The call options are $\langle \mathcal{A}_1, \neg$🐰$\rangle$, $\langle \mathcal{A}_1, \top \rangle$ and $\langle \mathcal{A}_2, \top \rangle$. Options $\langle \mathcal{A}_1, \neg$🐰$\rangle$ and $\langle \mathcal{A}_2, \top \rangle$ are applicable in the root's initial state. In each RM state, a metacontroller chooses an option with the aim of reaching an RM's accepting state as soon as possible. The formula option policies and metacontrollers are trained with Q-learning using a pseudo-reward function tailored to the formula and the RM's reward transition function, respectively.

An *option hierarchy* $\boldsymbol{\omega}$ manages the options currently executing. Initially, $\boldsymbol{\omega}$ is empty. At each step, the agent uses metacontrollers to add options to $\boldsymbol{\omega}$ until a formula option is added. In Figure 1b, the metacontroller in the root's initial state may choose option $\langle \mathcal{A}_2, \top \rangle$ followed by option 🐰, resulting in $\boldsymbol{\omega} = [\langle \mathcal{A}_2, \top \rangle,$ 🐰$]$, and actions are then selected according to option 🐰. After each action, formula option policies are updated and the new hierarchy state determines whether any option in $\boldsymbol{\omega}$ has terminated. A formula option terminates if the hierarchy state changes, while a call option terminates if it does not appear in the call stack of the new hierarchy state. Termination is applied bottom-up starting from the formula option, and stopped when an option does not terminate. Finally, the metacontrollers are updated for the terminated options, and call options that appear in the call stack of the new hierarchy state but not in $\boldsymbol{\omega}$ are added to $\boldsymbol{\omega}$ so that the corresponding metacontrollers can be updated later. We remark that the agent may not move through the hierarchy as intended: if the agent observes {🎋} while pursuing 🐰, it moves to RM $\mathcal{A}_1$ and not to $\mathcal{A}_2$ as originally intended.

[1] We assume reward functions are $r_i(u, u') = 1$ if $u \notin U_i^A$ and $u' \in U_i^A$, and 0 otherwise for all $0 \leq i < m$.

[2] These RMs are different from the original ones [8] in that (i) there are calls to other RMs in a hierarchy, (ii) there are explicit accepting and rejecting states, and (iii) transitions are given by propositional logic formulas over $\mathcal{P}$ instead of sets of propositions.

Learning the Hierarchies An HRM is automatically learned for each task in a set of composable tasks following a *curriculum learning* method [6]. Each task is assigned a level depending on its subtasks (e.g., Table 1 shows the levels for CraftWorld tasks), and learning progresses from lower to higher levels. Initially, level 1 tasks are chosen with equal probability while higher level tasks cannot be chosen. When task i terminates, its average return R_i is updated using the last undiscounted return r as $R_i \leftarrow \beta R_i + (1-\beta)r$, where β is a hyperparameter. The probability of choosing task i in the next episode is given by $c_i / \sum_k c_k$, where $c_i = 1 - R_i$ (the maximum return is assumed to be 1). When the minimum average return of tasks in the current level or lower surpasses a threshold Δ, the current level increases by 1.

Learning an HRM is analogous to previous work for flat machines [2]. Given a set of label traces, a set of propositions, a set of callable RMs and a number of RM states, an inductive logic programming system, ILASP, learns a state transition function that correctly recognizes the traces (e.g., goal traces finish in a root's accepting state). The callable RMs include all RMs in lower levels, and each RM has one accepting state and one rejecting state. To ease the induction, label traces are compressed (i.e., consecutive equal labels are merged into a single one), RMs are forced to be acyclic, DNFs consist of a single disjunct, and a symmetry breaking method is applied. The induction of HRMs is *interleaved* with policy learning: a new HRM is learned when an episode's label trace is not correctly recognized by the current HRM (e.g., a goal trace does not finish in the root's accepting state).

The first HRM is learned from a set of κ goal traces collected by exploring, similar to other works [9, 11]. For level 1 tasks, the agent performs a random walk, while in higher levels the agent uses HRMs (i.e., call options) and formulas (i.e., formula options) from lower levels. Enhancing exploration with options allows collecting goal traces faster, especially when labels are sparse. Finally, the κ_s shortest traces are used to learn the HRMs in order to reduce the running time.

3 Evaluation and Discussion

We evaluate our method in two domains. CraftWorld is a modification of MiniGrid [1] that adds new object types (one per proposition) and tasks. The grid is fully observable and can be of three types: an open plan 7×7 grid (OP) as in Figure 1a, an open plan 7×7 grid with a lava location (OPL), and a 13×13 four rooms (FR) [7]. OPL has an extra proposition for lava (🔥), which must always be avoided. WaterWorld [8] is a 2D box containing 12 balls of 6 colors (2 per color), moving at constant speed in a fixed direction. The agent ball can change its velocity in any cardinal direction. Propositions $\mathcal{P} = \{r, g, b, c, y, m\}$ denote ball colors. The agent observes the color of the balls it overlaps with. The tasks consist in observing specific sequences of colors [8]. Unlike CraftWorld, labels with multiple propositions are observable in WaterWorld, motivating the use of propositional formulas as an extra level of abstraction. Both domains are partially observable since the agent does not know the accomplished subgoals, which are encoded by the HRM.

Each experiment consists of 10 runs on a set of 10 random instances (e.g., by placing objects randomly in CraftWorld). In CraftWorld, OP and OPL have one object for each proposition, while FR has one or two. All experiments run for 100,000 episodes, each lasting a maximum of 300 steps. The curriculum has parameters $\beta = \Delta = 0.95$ and uses returns from the greedy policies evaluated in each task-instance pair every 100 episodes. ILASP has 2 hours to learn the HRMs for all tasks. We use $\kappa = 25$ for level 1 tasks, $\kappa = 150$ for level 2 tasks onwards, and $\kappa_s = 10$ for all tasks.

We define multiple DQNs at different levels of abstraction [5]. Each formula option and each RM is associated with a DDQN [10]. Metacontrollers provide Q-values for each option in an RM given an observation and an RM state (the output is masked according to the options available in the input RM state). We adopt ϵ-greedy exploration: each formula option and RM state is associated with its own ϵ, which is linearly annealed. For formula options, ϵ decreases after each step performed with that formula, while for metacontrollers ϵ decreases after having finished an option that started in that state. Formula option policies and metacontrollers are trained using different discount factors γ. Each RM has its own experience replay buffer, whereas all formula options share a common buffer (a form of intra-option learning [7]).

Figure 2 shows learning curves for the tasks of each domain, each measuring the undiscounted return obtained by the greedy policy every 100 episodes. The dotted vertical lines correspond to episodes where an HRM is learned. To ease visibility, some curves are not displayed for all training episodes. All tasks generalize across instances, and the curriculum is visible in all domains: when the return for tasks at a level is close to 1, the HRMs and policies in the next level start to be learned. In CraftWorld, learning in OP is easier than in OPL and FR. OPL is harder than OP because (1) dead-ends hinder observing goal examples for level 1 tasks using a random policy; (2) the root RMs must include rejecting states and use an extra proposition, complicating learning; (3) all non-lava policies must avoid the lava; (4) policies to reach the lava must be learned; and (5) metacontrollers must learn that edges labeled with 🔥 should not be chosen. Similar factors make FR harder than OP and OPL. The collection of goal examples in level 1 tasks is challenging with ϵ-greedy, especially those with several subgoals (e.g., Batter) where convergence is delayed relative to those with fewer subgoals. It is also more difficult to generalize across instances since FR's grid is bigger. In all cases, we observe that once level 1 tasks are mastered, learning in higher level tasks is fast owing to the reuse of lower level tasks' RMs.

The average running times (in seconds) of the HRM learning system across runs are 1257.8 (163.2) for OP, 1706.0 (302.8) for OPL, and 669.9 (113.1) for FR (standard errors in brackets). Including the lava (OPL) increases the time needed to learn

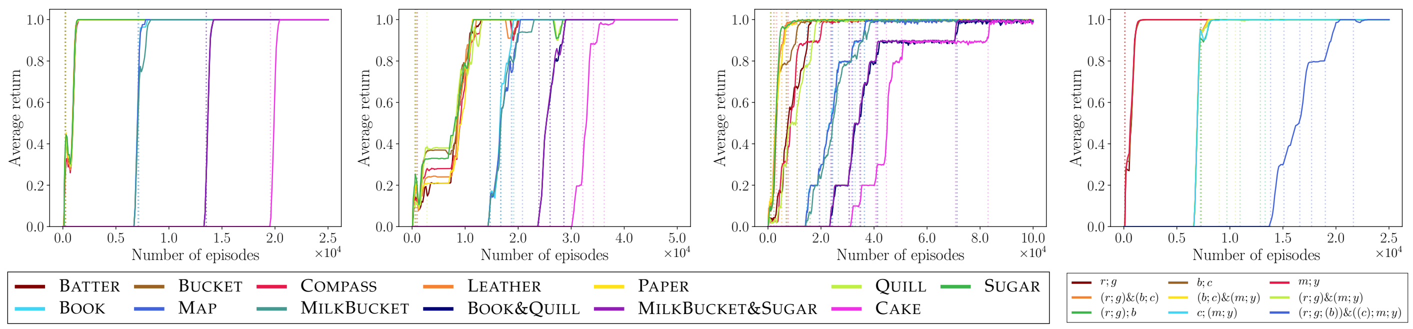

Figure 2: Learning curves for CraftWorld (OP, OPL, and FR) and WaterWorld.

the HRMs due to (1) a bigger hypothesis space caused by the increase in the number of propositions and the inclusion of rejecting states, and (2) the need to cover more example traces (OPL involves dead-end traces). In the case of FR, despite the learned HRMs are similar to those in OP, the running time is lower. This may be due to the example traces being shorter: propositions are sparsely distributed in FR and dense in OP. In all cases, around 90% of the running time is spent on the HRMs for Book, Map and Cake. Learning a multi-level HRM is less demanding than learning an equivalent flat one: the only task out of level 1 whose flat HRM can be learned within 2 hours is MilkBucket, which consists of 4 states. The flat HRM for Book, which is also a level 2 task, contains twice as many states (see Figure 1c). This shows that leveraging task compositionality helps learning RMs which could not have been learned in previous work.

The performance of exploration is evaluated by measuring for each task the number of episodes between the activation of its level and the learning of its first HRM. We compare the performance by using only primitive actions versus using call and formula options as well. Using only primitive actions leads to a higher number of episodes (i.e., it takes more time to collect the set of κ examples). While the Book task in the OP scenario needs 6957.2 (302.8) episodes using primitive actions, only 500.3 (9.6) episodes are required if options are also used. Delaying the learning of an HRM incurs a general delay since it takes longer to switch to higher levels. In the case of FR, observing goal examples is harder and surpassing level 2 never occurs if primitive actions are exclusively used. In addition, using sets of goal examples is shown to be convenient: using $\kappa = \kappa_s = 1$ in OP causes experiments to time out when on level 2 or higher in 9/10 runs. Finally, we evaluate policy learning alone by comparing the learning curves for handcrafted non-flat and equivalent flat HRMs. We observe both converge similarly in the simplest tasks, while non-flat HRMs speed up convergence in the hardest ones.

In future work, we plan to modify the curriculum learning component by removing the pre-established levels for each task, and allowing policies from lower levels to be used without being close to perfect.

References

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

[2] 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.

[3] M. Gaon and R. I. Brafman. Reinforcement Learning with Non-Markovian Rewards. In *AAAI*, 2020.

[4] M. Hasanbeig, N. Y. Jeppu, A. Abate, T. Melham, and D. Kroening. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. In *AAAI*, 2021.

[5] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *NeurIPS*, 2016.

[6] 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 *NeurIPS*, 2019.

[7] 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.

[8] 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 *ICML*, 2018.

[9] 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 *NeurIPS*, 2019.

[10] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-Learning. In *AAAI*, 2016.

[11] 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 *ICAPS*, 2020.