Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning (Supplementary Material)

Rodrigo Toro Icarte Toryn Q. Klassen Richard Valenzano Sheila A. McIlraith

A. Proof of Theorem 4.1

For every state of a reward machine in Σ, that is, for each $u_j^o \in \bigcup_{i=1}^{|\Sigma|} U^i$, QRM learns one q-value function. Let $\tilde{q}_j^o$ be the q-value function for state j in the reward machine o. Given any experience (s, a, s'), QRM updates the value of $\tilde{q}_j^o(s, a)$ using the following learning rule:

$$\tilde{q}_j^o(s, a) \xleftarrow{\alpha} r(s, a, s') + \gamma \max_{a'} \tilde{q}_k^o(s', a')$$

where $u_k^o = \delta_u^o(u_j^o, L(s'))$ and $r = \delta_r^o(u_j^o, u_k^o)$. This update is equivalent to a q-learning update over the cross-product MDP $\mathcal{M}_o$ with respect to the reward machine o (Observation 1). The reason is that whenever the agent performs action a in state $\langle s, u_j^o \rangle$ of $\mathcal{M}_o$ and gets to a state whose first component is s', the q-learning update is then necessarily over the experience $(\langle s, u_j^o \rangle, a, \langle s', u_k^o \rangle))$ using the reward function $\delta_r^o(u_j^o, u_k^o)$ as u_k^o is computed using $\delta_u^o(u_j^o, L(s'))$. As q-learning converges to the optimal q-function, then the optimal q-value function for $\tilde{q}_j^o$ is equivalent to the optimal q-function for $\tilde{q}(\langle \cdot, u_j^o \rangle, \cdot)$ in $\mathcal{M}_o$. As QRM selects actions according to the q-function of the current state machine, QRM converges to the same policy as solving $\mathcal{M}_o$ for every machine in Σ. Notice that convergence is guaranteed because QRM uses a stochastic exploratory policy (ϵ-greedy) which ensures trying every state-action pair infinitely often *in the limit*.

B. List of Tasks

An informal descriptions of the tasks that were used in the office domain, the minecraft domain, and the water world can be found in Table 1, Table 2, and Table 3, respectively. Their encoding as reward machines is available at `https://bitbucket.org/RToroIcarte/qrm`.

Table 1. Tasks for the office world. They are built over a set of propositional symbols $\{c, m, o, ✻, A, B, C, D\}$. Symbol c represents picking coffee; m, picking the mail; o, at the office; ✻, an office decoration; and A, B, C, and D are marked locations.

Task #	Description
1	deliver coffee c to the office o without breaking any decoration ✻
2	deliver mail m to the office o without breaking any decoration ✻
3	patrol locations A, B, C, and D, without breaking any decoration ✻
4	deliver a coffee c and the mail m to the office o without breaking any decoration ✻

Table 2. Tasks for the Minecraft domain. Each task is described as a sequence of events. They must occur in the order given, except as noted. There can be time between the occurrences of events of the sequence, and there are no constraints on what other events can occur between the ones of the sequence.

Task #	Task name	Description
1	make plank	get wood, use toolshed
2	make stick	get wood, use workbench
3	make cloth	get grass, use factory
4	make rope	get grass, use toolshed
5	make bridge	get iron, get wood, use factory (the iron and wood can be gotten in any order)
6	make bed	get wood, use toolshed, get grass, use workbench (the grass can be gotten at any time before using the workbench)
7	make axe	get wood, use workbench, get iron, use toolshed (the iron can be gotten at any time before using the toolshed)
8	make shears	get wood, use workbench, get iron, use workbench (the iron can be gotten at any time before using the workbench)
9	get gold	get iron, get wood, use factory, use bridge (the iron and wood can be gotten in any order)
10	get gem	get wood, use workbench, get iron, use toolshed, use axe (the iron can be gotten at any time before using the toolshed)

Table 3. Tasks for the water world. By "(color1 then color2)" we mean the task of eventually touching a ball of color1 and then eventually touching a ball of color2 (while possibly touching other balls along the way). A task like "(color1 then color2 then color3)" is similar but with three types of balls to touch. By "(subtask1) and (subtask2)" we mean that the agent must complete the tasks described by subtask1 and subtask2, but in any order (and possibly interleaved). By "(color1 strict-then color2)" we mean the task which is like "(color1 then color2)" but where the agent is not allowed to touch balls of other colors during execution.

Task #	Description
1	(red then green)
2	(blue then cyan)
3	(magenta then yellow)
4	(red then green) and (blue then cyan)
5	(blue then cyan) and (magenta then yellow)
6	(red then green) and (magenta then yellow)
7	(red then green) and (blue then cyan) and (magenta then yellow)
8	(red then green then blue) and (cyan then magenta then yellow)
9	(red strict-then green strict-then blue)
10	(cyan strict-then magenta strict-then yellow)