**Abstract.** This paper examines the problem of how to teach multiple tasks to a Reinforcement Learning (RL) agent. To this end, we use Linear Temporal Logic (LTL) as a language for specifying multiple tasks in a manner that supports the composition of learned skills. We also propose a novel algorithm that exploits LTL progression and off-policy RL to speed up learning without compromising convergence guarantees, and show that our method outperforms the state-of-the-art approach on randomly generated Minecraft-like grids.

**Running Example**

Luigi can collect raw materials:

- wood
- grass
- iron
- gold
- gems

... and make new objects:

- get wood
- get iron
- use the factory
- find tools
- make new objects

E.g., make a bridge: get wood, get iron, and use the factory

**Motivation**

**How do you describe a task to an RL agent?**

- **Task specification**
  - Reward function
  - Language

**Why would you want such a language?**

- To define new tasks faster.
- To transfer learning between tasks.

We use **Linear Temporal Logic (LTL)** to specify tasks and **LPOPL** to transfer learning between multiple tasks.

**Related Work**

**Exemplar task**

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>get wood</td>
<td>✓</td>
</tr>
<tr>
<td>get wood and then use the factory</td>
<td>✓</td>
</tr>
<tr>
<td>get wood or iron</td>
<td>✓</td>
</tr>
<tr>
<td>get grass and iron</td>
<td>✓</td>
</tr>
<tr>
<td>do not leave the shelter at night</td>
<td>✓</td>
</tr>
<tr>
<td>Off-policy learning</td>
<td>✓</td>
</tr>
<tr>
<td>Task decomposition</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Previous works using variants of LTL in RL** (e.g. [4, 5, 3]) do not exploit task decomposition or off-policy RL.

**Specifying Tasks in LTL**

Given a set of high-level events $\mathcal{E}$ that the RL agent can detect, such as $\mathcal{E} = \{\text{get wood}, \text{get iron}, \text{get grass}, \text{used workbench}, \text{used factory}, \text{is night}, \text{at shelter}\}$, we can use LTL to define tasks by composing occurrences of events in $\mathcal{E}$. LTL augments propositional logic with temporal operators $\bigcirc$ (next), $\diamond$ (eventually), and $U$ (until):

$$\psi := p | \psi \land \psi | \bigcirc \psi | \diamond \psi | U \psi \quad | \psi \in \mathcal{E}$$

**Examples:**

- eventually get wood
- eventually (get grass and eventually used factory)
- eventually get wood or eventually get iron
- eventually get grass and eventually get iron
- (is night $\land$ at shelter) until get wood

**Experiments**

**Goals:** study LPOPL $+ $ DQN; compare with standard RL and with alternative decomposition methods

**Baselines**

- DQN-L
- HRL-E
- Meta-Controller

**Conclusion**

- LPOPL takes tasks defined with LTL, decomposes them using LTL progression, and learns the subtasks.
- LPOPL outperformed DQN and HRL over various tasks.
- See https://bitbucket.org/RToroIcarte/lpopl for details.

**References**