# Teaching Multiple Tasks to an RL Agent using LTL

Reasoning Agents

Amr Aly
Joris Demiraj
Ahmed El Sheikh

La Sapienza Universita Di Roma

A. Y. 2019 - 2020

## Table of contents

## Outline

## Reinforcement Learning Framework

– Markov Decision Processes (MDPs)

$$M = \langle S, A, T, R, \gamma \rangle,\ R : S \times A \times S \to Pr(\mathbb{R})$$

$Q_\pi(s, a)$ is the expected discounted return of selecting action $a$ in state $s$ and then selecting actions according to policy $\pi$.

– Q-Learning
Off-policy RL methods learn a target policy while using some other behaviour policy for action selection. After an interaction, the Q-value is updated as follows,

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q'(s', a') - Q(s, a)]$$

## LTL

Syntax: Propositional symbols, boolean and temporal operators

$$\varphi ::= p \mid T \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \bigcup \varphi_2$$

Semantics:
Relative to a sequence of truth assignments, $\sigma = \langle \sigma_0, \sigma_1, .. \rangle$, $p \in \mathcal{P}$

- $\langle \sigma, i \rangle \models p$ iff $p \in \sigma_i$
- $\langle \sigma, i \rangle \models \neg\varphi$ iff $\langle \sigma, i \rangle \not\models \varphi$
- $\langle \sigma, i \rangle \models \varphi_1 \wedge \varphi_2$ iff $\langle \sigma, i \rangle \models \varphi_1$ and $\langle \sigma, i \rangle \models \varphi_2$
- $\langle \sigma, i \rangle \models \bigcirc\varphi$ iff $\langle \sigma, i+1 \rangle \models \varphi$
- $\langle \sigma, i \rangle \models \varphi_1 \bigcup \varphi_2$ iff there exists $j$ such that $i \leq j$ and $\langle \sigma, j \rangle \models \varphi_2$, and for all $k$ such that $i \leq k < j$, $\langle \sigma, k \rangle \models \varphi_1$

## Outline

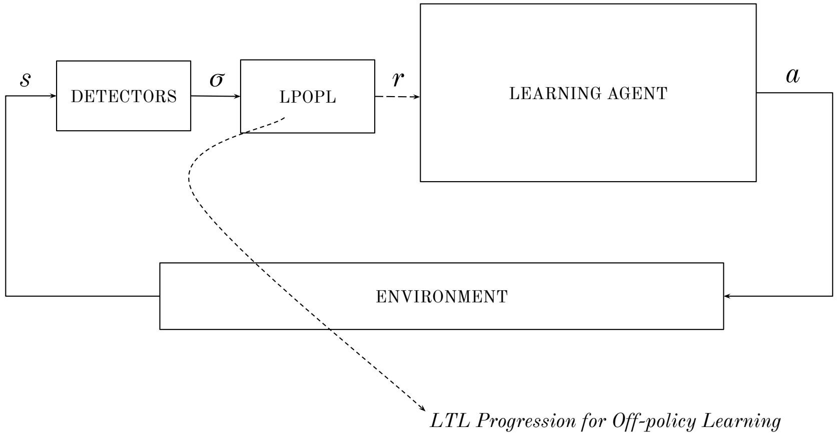
## Introduction

Goal: Teach an RL to perform multiple tasks using LTL for task specification

Why LTL?

1. Well-defined semantics and thus unambiguously interpretable
2. Easy to define rewards for complex and temporally extended tasks
3. Able to have meaningful state representations (through predefined – domain-specific – properties and events)
4. Global optimization of all tasks simultaneously (in contrast to hierarchical RL)

## Overview


$s$
DETECTORS
$\sigma$
LPOPL
$r$
LEARNING AGENT
$a$
ENVIRONMENT
*LTL Progression for Off-policy Learning*

## Problem Formulation

Task specification uses co-safe LTL. Co-safe LTL is the subset of LTL for which the truth of formulae can be ensured after a finite number of steps.

Examples:

1. $\Diamond p$: is co-safe, because once $p$ has been satisfied, what happens afterwards is irrelevant.
2. $\neg\Diamond q$: is not co-safe, because it can only be satisfied if $q$ is never true over an infinite number of steps. (Solution: $\neg q \bigcup p$)

## Problem Formulation

The agent receives a reward $(+1)$ iff a goal formula $\varphi$ is satisfied by the sequence of states visited in the episode, $(0)$ otherwise. Therefore, it follows that the problem is defined formally as a non-Markovian reward decision process (NMRDP).

$$N = \langle S, A, T, R, \gamma \rangle,\ R : S^* \to Pr(\mathbb{R})$$

and the $Q$-value function of a policy $\pi$ can be defined over sequences of states

$$Q_\pi(\langle s_0, .., s_t \rangle, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R(\langle s_0, .., s_{t+k+1} \rangle) | A_t = a \right]$$

## Problem Formulation

Classical RL methods assume the Markovian property, therefore, we move the history tracking from the process to the LTL specification itself. We define a multi-task co-safe LTL specification (MTCLS)

$$\tau = \langle S, A, T, \mathcal{P}, L, \Phi, \gamma \rangle = \{\langle S, A, T, R_\varphi, \gamma \rangle : \varphi \in \Phi\}$$

where $S, A, T$ and $\gamma$ are defined as in an MDP or NMRDP, $\mathcal{P}$ is a set of propositional symbols, $L : S \to 2^{\mathcal{P}}$ is the labelling function, and $\Phi$, the set of tasks, is a finite non-empty set of co-safe LTL formulae over $\mathcal{P}$.

$$R_\varphi(\langle s_0, .., s_n \rangle) = \begin{cases} 1 & \sigma_{0:n-1} \not\models \varphi \text{ and } \sigma_{0:n} \models \varphi \\ 0 & \textit{otherwise} \end{cases}$$

where $\sigma_{i:j} = \langle \sigma_i, .., \sigma_j \rangle = \langle L(s_i), .., L(s_j) \rangle$

## Algorithm – Overview

```
**Function** LPOPL($\gamma, \Phi, L, \mathcal{P}, N$)
    $t \leftarrow 0, i \leftarrow 0$;
    $\Phi^+ =$ ExtractSubtasks($\Phi, \mathcal{P}$);
    $Q \leftarrow$ InitializeQValueFunctions($\Phi^+$);
    **while** $t < N$ **do**
        $\varphi \leftarrow$ GetEpisodeTask($\Phi$);
        $t \leftarrow t +$ RunEpisode($Q, \varphi, L, \gamma, N - t$);
    **return** $Q^+$;
```

## Algorithm – Progression

1. $\text{prog}(\sigma_i, p) = \mathit{true}$ if $p \in \sigma_i$, where $p \in \mathcal{P}$
2. $\text{prog}(\sigma_i, p) = \mathit{false}$ if $p \notin \sigma_i$, where $p \in \mathcal{P}$
3. $\text{prog}(\sigma_i, \neg\varphi) = \neg\ \text{prog}(\sigma_i, \varphi)$
4. $\text{prog}(\sigma_i, \varphi_1 \wedge \varphi_2) = \text{prog}(\sigma_i, \varphi_1) \wedge \text{prog}(\sigma_i, \varphi_2)$
5. $\text{prog}(\sigma_i, \bigcirc\varphi) = \varphi$
6. $\text{prog}(\sigma_i, \varphi_1 \bigcup \varphi_2) = \text{prog}(\sigma_i, \varphi_2) \vee (\text{prog}(\sigma_i, \varphi_1) \wedge \varphi_1 \bigcup \varphi_2)$

Let
$\Phi = \{\Diamond(p \wedge \bigcirc\Diamond q),\ \Diamond(k \wedge \bigcirc\Diamond q)\}$

Ex: $\varphi_1 = \Diamond(p \wedge \bigcirc\Diamond q)$

$\text{prog}(\varphi_1)$
$\equiv \text{prog}(\mathit{true} \bigcup (p \wedge \bigcirc\Diamond q))$
$= \text{prog}(p \wedge \bigcirc\Diamond q) \vee \ldots$
$= \text{prog}(p) \wedge \text{prog}(\bigcirc\Diamond q)$
$= \mathit{true} \wedge \Diamond q$
$= \Diamond q$

## Algorithm – Subtasks

```
Function ExtractSubtasks($\Phi, \mathcal{P}$)
    $\Phi^+ \leftarrow \Phi$;
    repeat
        $\Phi_{\text{last}} \leftarrow \Phi^+$;
        for $\langle \tau, \varphi \rangle \in (2^{\mathcal{P}} \times \Phi_{\text{last}})$ do
            $\varphi' = \text{prog}(\tau, \varphi)$;
            if $\varphi' \notin \{\text{true}, \text{false}\}$ then
                $\Phi^+ \leftarrow \Phi^+ \cup \{\varphi'\}$;
    until $\Phi^+ = \Phi_{\text{last}}$;
    return $\Phi^+$;
```

## Algorithm – Q-Learning

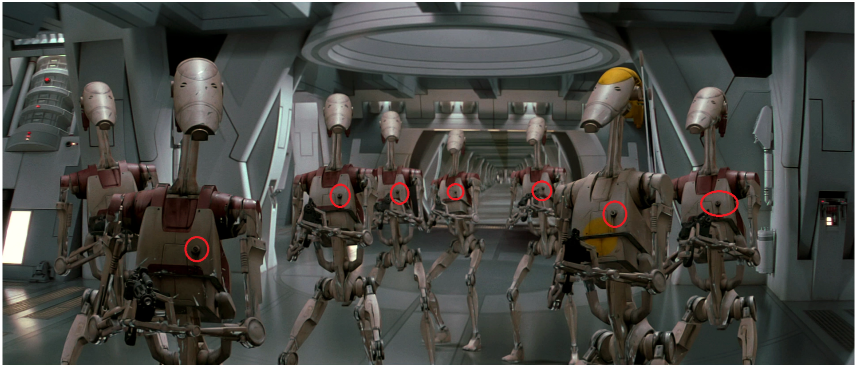
```
Function RunEpisode($Q, \varphi, L, \gamma, N$)
    $t \leftarrow 0$; $s \leftarrow$ GetInitialState();
    while $t < N$ do
        $\varphi \leftarrow \text{prog}(L(s), \varphi)$ ;
        if $\varphi \in \{\text{true}, \text{false}\}$ or EnvDeadEnd($s$) then
            break;
        $a \leftarrow$ GetActionEpsilonGreedy($Q_\varphi, s$);
        $s' \leftarrow$ EnvExecuteAction($s, a$);
        for $Q_\psi \in Q$ do
            $r \leftarrow 0$;
            $\psi' \leftarrow \text{prog}(L(s'), \psi)$;
            if $\psi' = \text{true}$ then
                $r \leftarrow 1$;
            if $\psi' \in \{\text{true}, \text{false}\}$ or EnvDeadEnd($s'$) then
                $Q_\psi(s,a) \leftarrow Q_\psi(s,a) + \alpha\left(r - Q_\psi(s,a)\right)$;
            else
                $Q_\psi(s,a) \leftarrow Q_\psi(s,a) +$
                    $\alpha\left(r + \gamma \max_{a'} Q_{\psi'}(s',a') - Q_\psi(s,a)\right)$;
        $s \leftarrow s'$; $t \leftarrow t + 1$;
    return $t$;
```

$$Q = \{ Q_{\Diamond(p \wedge \bigcirc \Diamond q)}, Q_{\Diamond(k \wedge \bigcirc \Diamond q)}, Q_{\Diamond q} \}$$

## Outline

## Introduction

What is a Restraining Bolts?



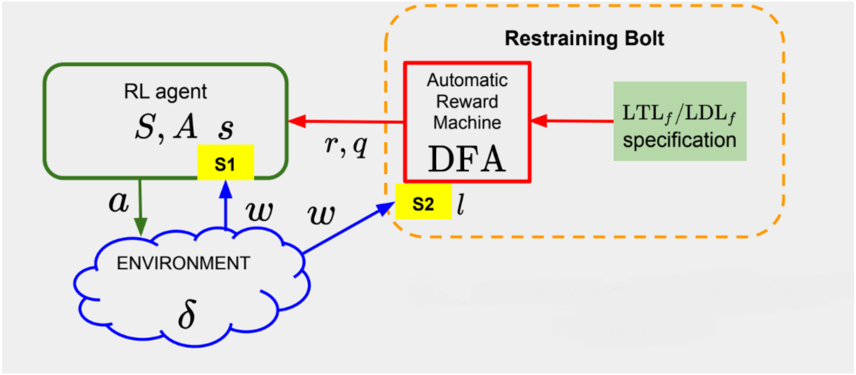
## Introduction

A Reinforcement Learning problem with $LTL_f/LDL_f$ restraining specifications is a pair $M_{ag}^{rb} = \langle M_{ag}, RB \rangle$, where:
$M_{ag} = \langle S, A, Tr_{ag}, R_{ag} \rangle$ is a learning agent with $Tr_{ag}$ and $R_{ag}$ hidden, and $RB = \langle \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m} \rangle$ is a restraining bolt formed by a set of $LTL_f/LDL_f$ formulas $\varphi_i$ over $\mathcal{L}$ with associated rewards $r_i$.

Solution: Policy $p : (Q_1 \times ... \times Q_m \times S)^* \rightarrow A$ maximizing $\mathbb{R}$

Goal: learn to act while conforming with $LTL_f/LDL_f$ specifications as much as possible.

## Representation



## Problem Formulation

1. The agent receives rewards based on $R_{ag}$ and the pairs $(\varphi_i; r_i)$.
2. Handle tasks in episodic nature.
3. When the episode ends and a new episode is started, a new trace is generated on which $LTL_f/LDL_f$ formulas are evaluated again.
4. Both $S$ and $\mathcal{L}$ are features' configurations but that capture different facets of the world.
5. Transform each $\varphi_i$ into DFA like $A_{\varphi_i} = \langle 2^{\mathcal{L}}, Q_i, q_{io}, \delta_i, F_i \rangle$ over fluents evaluations $\mathcal{L}$ with states $Q_i$ and final states $F_i$
6. Perform classical RL over a new MDP $M' = \langle Q_1 \times ... \times Q_m \times S, A, Tr'_{ag}, R' ag \rangle$
7. The optimal policy $\rho'_{ag}$ learned for M' is an optimal policy of the original problem.

## Implementation

Assume:

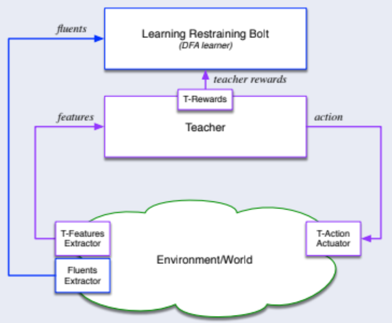
1. Learning phase in simulation
2. Execution phase on real world

Since all examples are of episodic nature, the learning phase is managed by an execution system that resets episodes when any of the following conditions is verified:

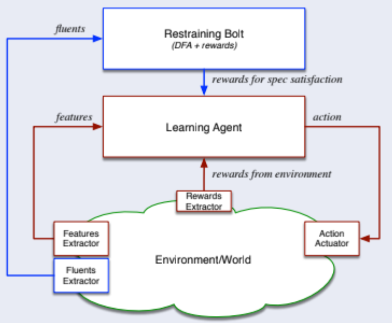
1. a state of the DFA where the formula is satisfied is reached,
2. a failure state (i.e., a state from which it is not possible to satisfy any formula) of the DFA is reached,
3. a maximum number of actions have been executed (to avoid infinite loops).

## Implementation

### Learn Restraining Bolt



### Use Restraining Bolt



## Outline

## Theoretical Differences

Reusability:

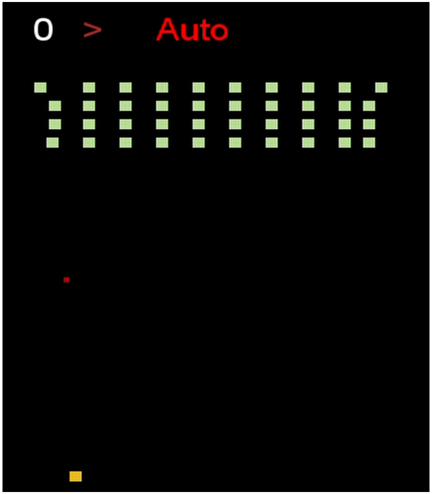
- LPOPL uses domain-specific vocabulary which renders the defined tasks (formulae) unusable in different environments
- The Restraining Bolt is issued by a different manufacturer from the agent which means it can be used on different agents

Adding new restraints:

- In LPOPL we need to define extra vocabulary with extra exponential overhead on the overall system
- In RB we need to define a new specification bolt which need not be related to the previous restraints

## Example 1 – Space Invaders

- Goal: Kill all aliens without being killed or overrun
- Deterministic environment
- State Space: x-position of shooter
- Action Space: $\{MOVE, FIRE\}$
- Extra specification: Prioritize shooting more dangerous aliens that exceed a certain threshold



## Example 1 – Space Invaders

- Restraining Bolt:

$$\varphi_{RB} = \Box\left(\left(y = min(Y) \leq threshold\right) \rightarrow \bigcirc\left(\neg A_{x,y} \wedge x = min(d(s,X))\right)\right)$$

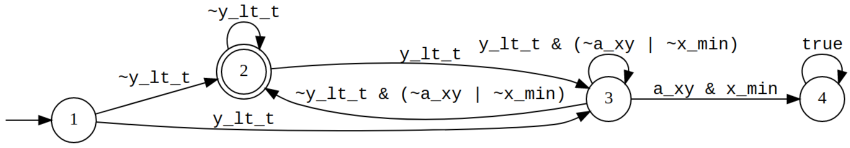
- Co-Safe LTL:

$$\varphi_{CSL} = \left(\left(y = min(Y) \leq threshold\right) \rightarrow \bigcirc\left(\neg A_{x,y} \wedge x = min(d(s,X))\right)\right)$$
$$\bigcup \forall x \forall y (\neg A_{x,y})$$

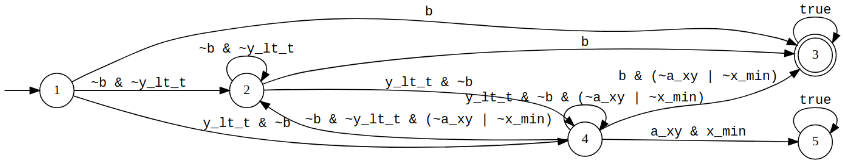
- $Y$: set of y-positions of each row
- $X$: set of x-positions of aliens in row $y = min(Y)$
- $A_{x,y}$: indicates whether an alien is present in location $(x, y)$
- $d(s, X)$: set of distance between $s$ and every element in $X$
- $min(X)$: minimum value of set $X$

## Example 1 – Space Invaders

- RB DFA:



- Co-Safe LTL DFA: (*Note:* $b = \forall x \forall y (\neg A_{x,y})$ "all aliens are dead")



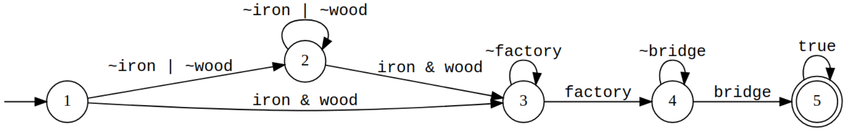
## Example 2 – Minecraft

- Goal: Cross a river by making a bridge (by collecting wood and iron first, and using the factory) and use the bridge afterwards, to finally get the gold.
- Deterministic environment
- Propositions $\mathcal{P} = \{got\_wood, got\_iron, use\_factory, ...\}$
- Restraining Bolt:
  $\varphi = \Diamond(got\_iron \wedge got\_wood) \wedge \bigcirc\Diamond use\_factory \wedge \bigcirc\Diamond use\_bridge$

  (*the formula is also Co-Safe*)

## Example 2 – Minecraft

DFA (for both RB and CSL):



## Outline

## Conclusion

- LPOPL is not efficient for big environments (in terms of propositions) due to its exponential overhead
- Both methods guarantee "probabilistic" satisfiability. That is if the goal formula is satisfiable, both methods will converge to an optimal policy. (model checking required)

## References

- De Giacomo, Giuseppe, and Moshe Y. Vardi. "Linear temporal logic and linear dynamic logic on finite traces." Twenty-Third International Joint Conference on Artificial Intelligence. 2013.
- De Giacomo, Giuseppe, et al. "Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications." Proceedings of the International Conference on Automated Planning and Scheduling. Vol. 29. No. 1. 2019.
- Toro Icarte, Rodrigo, et al. "Teaching multiple tasks to an RL agent using LTL." Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2018.
- Fahiem Bacchus and Froduald Kabanza. 2000. Using temporal logics to express search control knowledge for planning. Artificial Intelligence 116, 1-2 (2000), 123–191.

## References

- Faruq, Fatma, et al. "Simultaneous task allocation and planning under uncertainty." 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018.
- LTL2DFA DOI: 10.5281/zenodo.2598764