# Reinforcement Learning for $\mathrm{LTL}_f/\mathrm{LDL}_f$: Theory and Implementation

Marco Favorito

M.Sc. in
Engineering in Computer Science
at Sapienza, University of Rome

A.Y. 2017/2018

SAPIENZA
UNIVERSITÀ DI ROMA

# Introduction

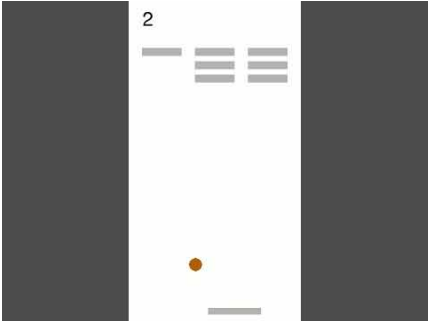
- Classic Reinforcement Learning:
  - An *agent* interacts with an *environment* by taking *actions* so to maximize *rewards*;
  - No knowledge about the transition model, but assume Markov property (history does not matter): Markov Decision Process (MDP)
  - Solution: (Markovian) policy $\rho : S \rightarrow A$

- RL for Non-Markovian Decision Process (NMRDP):
  - Rewards depend from history, not just the last transition;
  - Specify proper behaviours by using temporal logic formulas;
  - Solution: (Non-Markovian) policy $\rho : S^* \rightarrow A$
  - Reduce the problem to MDP (with extended state space)

- In (Brafman et al. 2018) specify reward using:
  - Linear-time Temporal Logic on Finite Traces $\mathrm{LTL}_f$
  - Linear-time Dynamic Logic on Finite Traces $\mathrm{LDL}_f$

## Example of NMRDP: BREAKOUT

- Non-Markovian reward: remove columns from left to right
- The solution space is restricted
- Policy must depend from a sequence of states: $\rho : S^* \to A$



# Goals of the thesis

- Theoretical foundations for RL over NMRDP with $\mathrm{LTL}_f/\mathrm{LDL}_f$ rewards by leveraging (Brafman et al. 2018)
- Formalization and solution of a new problem: RL for $\mathrm{LTL}_f/\mathrm{LDL}_f$ goals
  - two-fold representation of the world
    - Low-level, used by the learning agent
    - High-level, used to specify the goal formulas
- Deal with sparse rewards and design a way to improve exploration
- Implementation of the topics described above

# $\text{LTL}_f$ and $\text{LDL}_f$ (De Giacomo and Vardi, 2013)

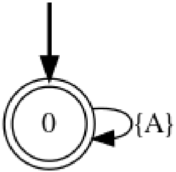
- Linear Temporal Logic on finite traces: $\text{LTL}_f$
  - Exactly the same syntax of LTL
  - Interpreted over finite traces
    - Next: $\bigcirc \mathit{happy}$
    - Until: $\mathit{reply}\,\mathcal{U}\,\mathit{acknowledge}$
    - Eventually: $\Diamond \mathit{rich}$
    - Always: $\Box \mathit{safe}$
- Linear Dynamic Logic on finite traces: $\text{LTL}_f$
  - Merging of $\text{LTL}_f$ and Regular Expressions
    - $[\mathit{true}^*](\mathit{safe})$
    - $\langle A^*; B\rangle \mathit{End}$
- Reasoning in $\text{LTL}_f/\text{LDL}_f$:
  - transform formulas $\varphi$ into NFAs or DFAs $\mathcal{A}_\varphi$
  - For every trace $\pi$ and $\text{LTL}_f/\text{LDL}_f$ formula $\varphi$:

$$\pi \models \varphi \iff \pi \in \mathcal{L}(\mathcal{A}_\varphi)$$

## From $\text{LTL}_f/\text{LDL}_f$ formulas to automata: examples

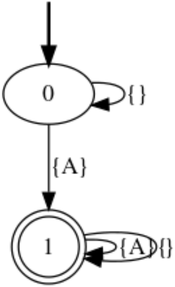
E.g. For $\pi = \langle \{\}, \{A\}, \{A, B\} \rangle$,

$\pi \not\models \Box A$ $\qquad$ $\pi \models \Diamond A$ $\qquad$ $\pi \not\models \langle (A; B)^* \rangle End$



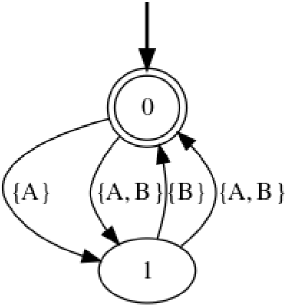

$\Box A$

*Always A*




$\Diamond A$

*Eventually A*




$\langle (A; B)^* \rangle End$

*Alternating A and B*

# FLLOAT: From $\text{LTL}_f/\text{LDL}_f$ tO AutomaTa

- Python package supporting:
  - $\text{LTL}_f/\text{LDL}_f$ formulas: parsing, syntax and semantics
  - Translation to automata (NFA, DFA, on-the-fly DFA)
- $\Box A$:

```
from flloat.parser.ltlf import LTLfParser
parser = LTLfParser()
always_A = parser("G(A)")
dfa = always_A.to_automaton(determinize=True)
dfa.to_dot("always_A.svg")
```

- $\langle (A; B)^* \rangle End$:

```
from flloat.parser.ldlf import LDLfParser
parser = LDLfParser()
alternating_AB = parser("<(A;B)*>end")
dfa = alternating_AB.to_automaton(determinize=True)
dfa.to_dot("alternating_AB.svg")
```

# RL for NMRDP with $\text{LTL}_f/\text{LDL}_f$ rewards

- Given an NMRDP $\mathcal{N}$ with:
  - rewards specified by $\text{LTL}_f/\text{LDL}_f$ formulas
  - over a set of propositional symbols $\mathcal{P}$
  - agent's state space: $S \subseteq 2^{\mathcal{P}}$
- We can transform it into an equivalent MDP $\mathcal{M}$ (Brafman et al. 2018)
  - the state space of $\mathcal{M}$ is extended:

$$S' = S \times Q_1 \times \cdots \times Q_m$$

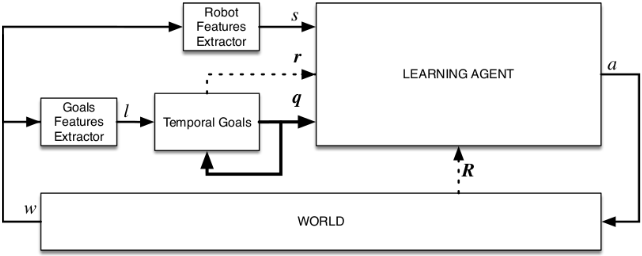
    where $Q_i$ is the set of states of $\mathcal{A}_{\varphi_i}$
  - reward is given iff $q_i$ is an accepting state of $\mathcal{A}_{\varphi_i}$
- An optimal policy for $\mathcal{M}$ is optimal for $\mathcal{N}$
- We reduced RL for $\mathcal{M}$ to RL for $\mathcal{N}$

# RL for $\mathrm{LTL}_f/\mathrm{LDL}_f$ goals: a new problem

**Two-fold representation** of the world $\mathcal{W}$:

- An agent learning an MDP with **low-level features** $S$, trying to optimize reward $R$
- $\mathrm{LTL}_f/\mathrm{LDL}_f$ goals $\{(\varphi_i, r_i)_{i=1}^m\}$ over a set of **high-level features** $\mathcal{F}$, yielding a set of fluents configurations $\mathcal{L} = 2^{\mathcal{F}}$

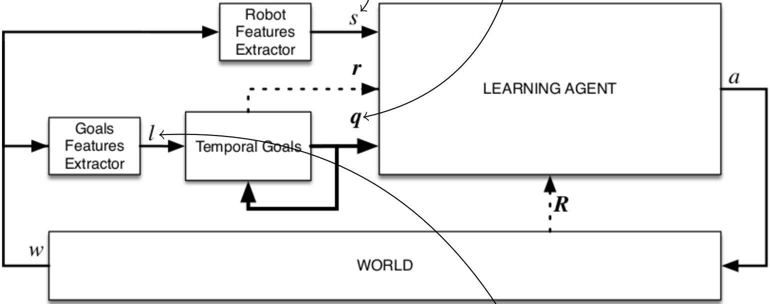
**Solution**: a non-Markovian policy $\rho : S^* \rightarrow A$ that is optimal wrt rewards $r_i$ and $R$.



Our approach:

- Transform each $\varphi_i$ into DFA $\mathcal{A}_{\varphi_i}$
- Do RL over an MDP $\mathcal{M}'$ with a transformed state space:

$$S' = \boxed{S} \times \boxed{Q_1 \times \cdots \times Q_m}$$




Notice: **the agent ignores the fluents** $\boxed{\mathcal{L}}$!
The actual RL relies on standard RL algorithms (e.g. Sarsa($\lambda$))

# Automata-based Reward Shaping

- *Reward sparsity* is the main issue: **Hard to learn complex behaviors without heuristics**

**Idea**: Give additional reward if, after a transition on $\mathcal{A}_\varphi$, we are closer to an accepting state (anticipate rewards)
Based on *Potential-Based Reward Shaping* (Ng et al., 1999):

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s)$$

Two modes:

- *Off-line*: $\Phi(q)$ inversely proportional to the distance from any accepting state of $\mathcal{A}_\varphi$
- *On-line*: $\mathcal{A}_\varphi$ and $\Phi$ are built from scratch and updated while learning and discovering new states/transitions. Using *Dynamic PBRS* (Devlin, 2012)

## RLTG (Reinforcement Learning for Temporal Goals)

Reinforcement Learning Python framework that implements our approach

- Depends on `flloat` for the construction of $\mathcal{A}_{\varphi_i}$
- Works also for classic RL
- Highly customizable, uses OpenAI Gym interface

Example (in pseudocode):

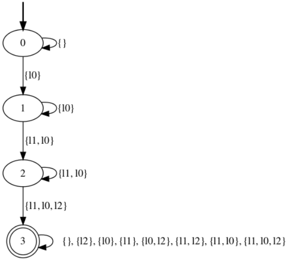
```
env = GymEnvironment()          # a Gym environment
agent = TGAgent(                # the "temporal goal" agent
    FeatureExtractor(...),      # generates the agent's space
    Brain(...), #abstraction of RL algorithms (e.g. Sarsa)
    [TemporalGoal(φ, r), ...] # list of temp. goal managers
)
trainer = TGTrainer(env, agent, stop_conditions=..., ...)
trainer.main(...)               # starts the learning process
```

# Experiments: BREAKOUT

BREAKOUT: remove columns/rows in a given order

- **low-level features**: paddle position, ball speed/position;
- **high-level features**: bricks status (broken/not broken)
- LTL$_f$/LDL$_f$ **goal** ($l_i$ means: the $i_{th}$ line has been removed.):

$$\langle(\neg l_0 \wedge \neg l_1 \wedge \neg l_2)^*; (l_0 \wedge \neg l_1 \wedge \neg l_2); (l_0 \wedge \neg l_1 \wedge \neg l_2)^*; \ldots; (l_0 \wedge l_1 \wedge l_2)\rangle tt$$

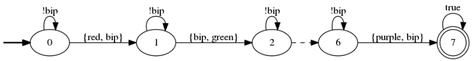


# Experiments: SAPIENTINO

SAPIENTINO: make a *bip* in each color, in a given order

- **low-level features**: robot position $(x, y)$;
- **high-level features**: color of current cell, bip-last-action
- $\text{LTL}_f/\text{LDL}_f$ **goal**:

$$\langle(\neg bip)^*; red \wedge bip; (\neg bip)^*; green \wedge bip; \ldots\rangle tt$$

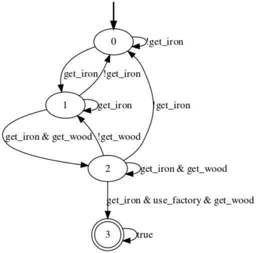


## Experiments: MINECRAFT

MINECRAFT: complete tasks by getting resources/using tools

- **low-level features**: robot position $(x, y)$;
- **high-level features**: *get_wood*, *use_workbench* etc.
- $\text{LTL}_f$/$\text{LDL}_f$ **goal**: three composite tasks like (approximately):

$$\langle true^* \rangle \langle get\_iron; get\_iron \wedge get\_wood; \ldots \wedge use\_factory \rangle tt$$



# Discussion

Why a two-fold representation for temporal goals?

- *Separation of concerns*: It allows to a better design of the RL system by improving modularity and flexibility;
- *Reduced state space*: It makes easier the learning by exploring a smaller state space and focusing only on relevant low-level features;
- *Simpler agent*: Move part of the complexity outside the agent.

Other observations:

- *Expressive Power*: $\mathrm{LDL}_f$ expressive as Monadic Second-Order logic (e.g. we can specify procedural constraints);
- No need of new algorithms, one can rely on off-the-shelf RL algorithms (Sarsa($\lambda$), Q-Learning, ...);
- The correlation between the two representations does not need to be formalized.

## Conclusions

Thesis results:

- Extended (Brafman et al. 2018) to the context of RL;
- Formalization of *RL for* $\text{LTL}_f/\text{LDL}_f$ *goals*, devising of a solution and analysis of the advantages;
- Provided an implementation and given experimental evidence of the goodness of our approach.

Future works:

- Minimal low-level representation to tackle $\text{LTL}_f/\text{LDL}_f$ goals;
- Optimize FLLOAT, enrich RLTG;
- Try the approach in several real world applications;
- Design of ad-hoc algorithms (e.g. "automata-aware" exploration);
- Extend the approach to the framework of Multi-Agent Systems.

Publication: https://arxiv.org/abs/1807.06333. Status: submitted