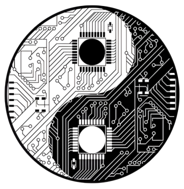
SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER, CONTROL AND MANAGEMENT ENGINEERING ANTONIO RUBERTI

Learning and Reasoning as *yin and yang* of future AI



PLANNING AND REASONING
MASTER PROGRAM IN ARTIFICIAL INTELLIGENCE AND ROBOTICS

Professor:
Paolo Liberatore

Student:
Lavalle Leonardo 1838492

Academic Year 2022/2023

Contents

1 Introduction

Artificial Intelligence (AI) is nowadays on everyone's lips. Everyone is talking about it, but nobody really knows what it is. The results achieved lately by AI, in particular neural networks, in fields such as computer vision and natural language processing are undeniable and impressive, leading to great excitement by both members of the artificial intelligence community and the broader public. Have we truly achieved human level intelligence? And more importantly, are we sure that we are going in the right direction?

AI seems to be the answer to everything, like a magical wand with the power to address every human problem. People are fascinated by it and scared of it at the same time. Why has this "magical power", assuming it exists, only appeared and gained recognition in recent years? Most of the algorithms and techniques employed now, were introduced and studied by AI researchers more than 50 years ago (e.g. the Multi Layer Perceptron, that is the ancestor of modern deep learning models, was developed in the 1960s). Everyone in the field says that the two main reasons of this "AI excitement" are the easy availability of massive amounts of data and the enhanced computational power at our disposal (GPUs, TPUs, etc.). Think about the huge number of open source datasets available online and web platforms like Google Colab which provide "free" GPUs for training our models. Neural networks success can also be linked to the increasingly more sophisticated statistical techniques for fitting functions, which are the key to their successful training such as backpropagation or dropout. But it's basically it! Adnan Darwiche, an AI researcher, in [1] talks about the importance of public perceptions about AI progress. He stresses out how a bad communication from the AI community can mislead people, create misperceptions and unwarranted fears, leaving much to the imagination. Let's look at the ChatGPT phenomenon: as far as it can be considered a "milestone" for AI history and an alien tool for the general audience (which is partly true), is nothing more that a very huge network (Large Language Model) which aims to learn the patterns and rules of language by fitting a function to a vast amounts of data. Nothing close to a breakthrough that justifies worrying about doomsday scenarios! Darwiche in [1] continue its discussion supporting that the tasks currently conquered by deep learning barely rise to the level of abilities possessed by many animals and that we are way far from a truly human level intelligence. By making this strong and provocative statement, he wants on one hand to lower people's expectations about AI (comforting them if they are somehow scared) and on the other to urge his colleagues to follow other directions with respect to function-based approaches.

We should start from AlphaGo success (2014). Why? Because it's not a deep neural network but its architecture is indeed based on a collection of AI techniques that have been in the books for at least fifty years: minimax technique, evaluation functions to cut off minimax search trees, stochastic search, learning from self play, reinforcement learning and in addition (only in addition) neural networks.

This is the path to follow. Integration of different AI techniques and paradigms is the key. As it happens in many other fields, we have to exploit each others strengths to help each others weaknesses. Recent successful AI systems, the so called function-based or "model-blind" systems, are *data hungry* and black boxes. They lack comprehensibility and is almost impossible to understand why decisions are made, creating challenges in placing trust in them. Instead, model-based systems, nearly forgotten by the majority of AI researcher (not by Sapienza ones), can help from this perspective since they inherently exhibit high explainability. This methods are related to that AI branch which is strictly based on mathematics, logic and formal methods: Knowledge Representation (KR) and Reasoning. This idea of "merging" is also supported by the conception of human intelligence as the integration of two fundamental cognitive abilities: *learning* (ability to learn from experience) and *reasoning* (ability to reason from what has been learned). The interplay and balance between these two aspects are in my opinion the way to go for having a great and positive impact on society: the *yin and yang of future AI.*

1.1 Sustainability and Explainability

Beyond the fact of the self legitimation nature of an intelligent system which has both the capability of learning from experience and reason about the acquired knowledge through it, we should investigate more why *knowledge* needs to be integrated in modern AI systems. I identified mainly two causes:
(i) The first one is more related to supervised learning and to all the methods which base their results on large amounts of data. It looks like deep neural networks success goes hand-in-hand with model's size and number of training data. In [2] is shown how the progress of AI systems is strongly reliant on increases in computing power. This voracious appetite for computing power and data "*is rapidly becoming*

economically, technically, and environmentally unsustainable". I can't imagine how many GPUs OpenAI, regarding ChatGPT training, has been exploited, how many hours of training has been performed, how much energy has been utilized and how much heat has been produced contributing to pollution and global warming. Since we cannot increase computational capacities indefinitely, as stated in [3], we need to employ more data efficient and knowledge-driven paradigms to create more robust AI. This general tendency to be sensitive to environmental issues (Sustainable AI) is demonstrated, for example, by the various commitments made by Meta during the release of its latest language model, LLaMA-2. They adopt eco-friendly energy sources in the training process and implemented tree-planting initiatives to counteract the adverse climatic effects.
(ii) Function-based methods are black boxes. The reasons behind neural networks actions or behind reinforcement learning policies are not clear and cannot be explicitly *explained*. We need tools to do so, and one way can be through knowledge integration. One simple example are Restraining Bolts, which are deeply explained in Section 4: users can induced a RL agent to behave coherently to an explicit temporal logic specification they defined and indeed being able to consciously control agent's behaviour. This can be very powerful, especially in a scenario where *safety* conditions must be met (see Section 4.1). For example, in medical diagnosis or autonomous driving, many neural systems cannot be employed and applied in the real world because of their non explainability nature. Another very important aspect that it's not covered in this essay regarding interpretability of AI, is *ethics*. In a world where AI will play an increasingly vital role in numerous pivotal aspects of daily life, policies and regulations will be (and are being) formulated. Administrations seek a deep understanding of how specific technologies operate, particularly the "why" behind their behaviors, while also desiring guarantees. Achieving these goals is made possible through the utilization of explainable AI, and knowledge-driven methods can significantly contribute to making this a reality.
In supervised learning, and in particular in neural models, the research direction is being pursued by Neuro-Symbolic AI (NeSy). What about Reinforcement Learning? Can be standard RL algorithms be improved by knowledge integration?

1.2 KR and RL

Combination of Knowledge Representation with Reinforcement Learning is a trend of which researchers are recently getting more and more excited about. The use of KR in support of RL can have many reasons. In my analysis, I've pinpointed three key motivations:

1. Reward functions are hard to express and design. Especially in real-world scenarios, where processes are non-markovian and engineering rewards becomes

very complicated. Micheal Littman, professor of Computer Science at Brown University, said at the IJCAI Conference in 2015 in Buenos Aires: "*Coming up with rewards in MDPs is too difficult! We need help from KR!*". That's why many works are aiming to model non-markovian rewards with the help of logic. This happens in Restraining Bolts (RBs) via *temporal logic* specifications or in other methods via *transducers* (see Section 5).

2. Reinforcement Learning usually does not scale up well to large problems. It typically takes a RL agent many trials until it can reach a satisfying policy. Indeed in some cases, we may want a more efficient learning, i.e. a faster learning of the optimal policy. This is exactly the purpose of the method outlined in Section 3 which employs LTL_f-based reward shaping. It also occurs in *preemptive* RBs because the agent doesn't explore unnecessary (unsafe) actions (see more details in Section 4.1) leading to a learning convergence improvement.

3. Urgency of providing *safety* guarantees to AI techniques based on learning (we already mention it in Section 1.1). RBs can be seen as a contribution in this direction, because they can be implemented to constrain the RL agent to satisfy certain safety conditions. The problem is that they don't guarantee the hard satisfaction of these constraints, and that's why *preemptive* RB (Section 4.1) have been recently proposed (2023).

In subsequent sections five papers about the topic are briefly presented. I chose them primarily for their ability to immediately convey to the reader the practical significance of knowledge integration in RL. Additionally, I wanted to pay tribute to our department and our university since we are at the forefront of this area of research (four out of five papers are from Sapienza researchers). The aim is not to give a precise technical overview of their content but to highlight their features, their strengths and their similarities between each other. In Section 2 there a few theoretical notions to let the reading of the report more fluent and clear. Sections 3, 4 and 5 contain papers overview and in Section 6 are reported the experiments I carried out on Restraining Bolts: both in their standard formulation and in an *Imitation Learning* (IL) setting (Section 4.2).

2 Background

As anticipated, in this section there are listed some theoretical preliminary notions that are required to not walk blind in the next sections of the report.

MDPs. A Markov Decision Process $\mathcal{M} = \langle S, A, Tr, R \rangle$ contains a set of S states, a set of A actions, a transition function $Tr : S \times A \rightarrow Prob(S)$, that returns for every state s and action a a distribution over the next state, and a reward function $R : S \times A \times S \to \mathbb{R}$ that specifies the reward received by the agent when transitioning form state s to state s' by applying action a. MDPs provide a mathematical framework for modeling decision-making problems where a RL agent interacts with an environment to achieve a final goal. The aim of the intelligent agent is to learn a particular function, the policy ρ, which assigns to each state the action to perform. After the training process, every MDP has an optimal policy ρ^*. The latter, can be either *Markovian*, where it depends only on the current state, or *non-Markovian*, where it may depend also on previous states. As we can imagine, designing non-Markovian reward functions and, consequently, compute non-Markovian policy functions is not trivial at all.

TD learning. Temporal Difference (TD) methods are a class of RL algorithms. They are used for estimating value functions and learning policies in MDPs. The key idea behind TD methods is that they update value estimates incrementally after each time step, making them more efficient than Monte Carlo ones. Without going into the details, we can mainly have two different TD algorithms: *(i)* Q-learning, which learns an optimal policy that maximizes the expected cumulative reward over time regardless of the policy the agent is currently following (*off-policy*); *(i)* SARSA, which learns the policy based on agent's current policy (*on-policy*). In all the experiments in Section 6, it's been employed the ϵ-greedy strategy. It means that, at training time, actions are chosen with a probability of ϵ accordingly to the current policy and with a probability of $1 - \epsilon$ randomly. This to handle the exploration-exploitation trade-off of the state space.
There exist a family of TD methods, $TD(\lambda)$, where λ parameter is used to control the trade-off between looking ahead to estimate future values and give instead credit to past states and actions. We then talk about Q(λ)-learning and SARSA(λ). When $\lambda = 0$, the algorithms are equal to the standard ones, while as $\lambda \to 1$ (as it was set up on all the experiments), it places more emphasis on giving credit to past state-action pairs. Like always, there's not a right answer to which value of λ to set, but it depends on the specific problem under examination.

LTL$_f$/LDL$_f$ The logic LTL_f is the classical linear time logic LTL interpreted over finite traces, formed by a finite sequence of propositional interpretations [4]. Formally, LTL_f formulas φ are defined as follows:

$$\varphi \quad ::= \quad \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \circ\,\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where ϕ is a propositional formula over the possible set of *fluents* (boolean propositions), $\circ$ is the *next* operator and $\mathcal{U}$ is the *until* operator. Notice that $\diamond\varphi$ (*eventually*) can be obtained as *true* $\mathcal{U}\varphi$ and that $\Box\varphi$ (*always*) can be obtained as $\neg\diamond\neg\varphi$. LDL_f is an extension of LTL_f. As explained in [4], it borrows the syntax from Propositional

Dynamic Logic (PDL) [5], but is interpreted over traces. The aim was to exploit the expressive power of regular expressions over finite traces RE_f with the capability of LTL_f formulas to express temporal specifications $[LDL_f = RE_f + LTL_f]$. More formally we can build LDL_f formulas φ as follows:

$$\begin{aligned} \varphi \quad &::= \quad tt \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\rho\rangle\varphi \\ \rho \quad &::= \quad \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1; \rho_2 \mid \rho^* \end{aligned}$$

in which tt stands for logical true; ϕ is still a propositional formula over the *fluents*; ρ represents a regular expression where '+' operator establishes that both ρ_1 and ρ_2 must hold, ';' operator determines that ρ_1 and ρ_2 must happen sequentially and '$*$' operator is for repetition; and '?' is the *test construct* from PDL [5] syntax. The most important way of building LDL_f formulas is through $\langle\rho\rangle\varphi$. Intuitively, it states that, from the current step in the trace, there exists an execution satisfying ρ such that its last step satisfies φ. To make it clearer, let's see the example of the Restraining Bolt specification in LDL_f used in the experiments concerning *Breakout* environment. In a nutshell (see Section 6 for more details), we have three *fluents*: c_1, c_2 and c_3. They represent the three columns of bricks present in the Breakout game. When a column is entirely broken, the relative fluent is satisfied. So, if we want to break the columns starting from left (c_1) to right (c_3), the LDL_f formula, according to the syntax just explained above, is:

$$<(\neg c_1 \wedge \neg c_2 \wedge \neg c_3)^*; c_1; (\neg c_1 \wedge \neg c_2 \wedge \neg c_3)^*; c_2; (\neg c_1 \wedge \neg c_2 \wedge \neg c_3)^*; c_3>tt$$

PPLTL. Pure Past Lineat-time temporal Logic (PPLTL) [6], is a variant of Linear-time Temporal Logic in which formulas are evaluated over finite traces and only past modalities are allowed. These kind of formulas are exploited by the *preemptive* RBs later explained in Section 4.1 by somewhat filtering out the *unsafe* actions the agent could perform. PPLTL syntax is defined as follows:

$$\varphi \quad ::= \quad \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \ominus \varphi \mid \varphi_1 \mathcal{S} \varphi_2$$

where ϕ is a propositional formula over the possible set of *fluents* (boolean propositions), $\ominus$ is the *yesterday* operator and $\mathcal{S}$ is the *since* operator. To be more precise: given a finite trace $\tau = \tau_0 \ldots \tau_n$ of propositional interpretations, $\ominus\varphi$ is true at τ_i if φ is true at τ_{i-1}, and $\varphi_1\mathcal{S}\varphi_2$ is true at τ_i if φ_2 is true at some τ_j ($j \leq i$) and φ_1 is true at $\tau_{j'}$ (for all j', $j < j' < i$).

Transducers. A finite-state *automaton* is a computational model that can read input strings in a given alphabet, it keeps track of its current state and it can produce output strings. Automata whose output are limited to a simple *yes* or *no* are called *acceptors*. We refer to them as *transducers* when they can produce strings of symbols as output. Two fundamental kinds of transducers are Moore machines [7] and Mealy machines [8]. In Section 5 it's shown how to use this tools to model non-markovian rewards.

3 Reward shaping

LTLf-based Reward Shaping for Reinforcement Learning paper [9], published in 2021 by Bruxelles University researchers, is the first work that made me approach this world of *knowledge* serving *learning* algorithms, in particular RL algorithms.
Reward shaping is a well-established method to incorporate domain knowledge in Reinforcement Learning allowing the RL agent to learn faster. The basic idea is to give small intermediate rewards to the algorithm that help it converge more quickly.

$$\boxed{Q(s,a) \leftarrow Q(s,a) + \alpha[r + \boxed{\underbrace{F(s,s')}_{\text{additional reward}}} + \gamma \max_{a'} Q(s',a') - Q(s,a)]}$$

Figure 1: This is reward shaping update rule in Q-learning algorithm. Notice the additional reward.

The additional reward is known as *domain knowledge*, i.e. notions the human engineers know in advance about the domain. Since reward shaping can, in general, not reach the optimal policy, we need some theoretical guarantees. These are given if the additional reward is in the form of:

$$F(s,s') = \gamma\phi(s') - \phi(s)$$

where $\phi(s)$ is the potential of state s and ϕ is some heuristic measure of the value of each state. In this case we talk about Potential-based Reward Shaping (PBRS) and we assure that optimal policy doesn't change.

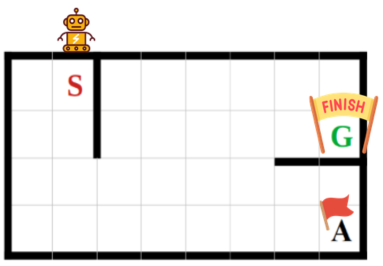


Figure 2: An example of flag collection problem where the start position (S), the flag (A) and the goal position (G).

The authors proposed a novel methodology that automatically generates reward shaping functions from user-provided Linear Temporal Logic formulas. This means that the domain knowledge is incorporated in the RL agent through LTL_f formulas which are then used to generate potential-based reward shaping. This type of temporal logic formulas, being quite similar to natural language, are a rich and compact way to express domain knowledge with minimum effort and indeed an easy manner to describe

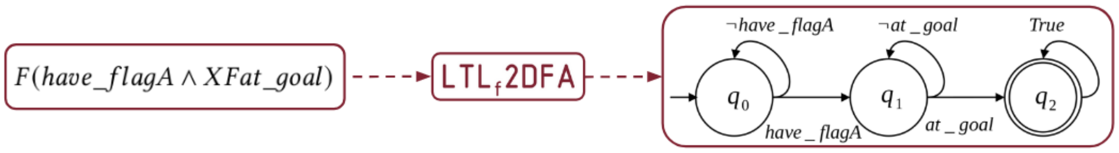


Figure 3: LTL_f formula and DFA for the *flag collection* environment in Figure 2. DFA could be generated by LTL$_f$2DFA Sapienza tool.

agent's goal. The LTL_f formula is then transformed to a DFA in order to track its satisfaction stage during the learning process and build the potential-based function.

As a running example to clarify their method, a simple *flag collection* problem is employed. The agent must simply collect a flag and then reach the goal (see Figure 2). In this scenario the user-knowledge to convey to the environment is: "Finally collect flag A and then finally be at the goal position". Both the corresponding LTL_f formula and the resulted DFA can be seen in Figure 3.
Without going too much into the details, we now show how they built function $\phi_t(s_t)$ (the *potential function* previously introduced) at time t and in MDP state s_t:

$$\phi_t(s_t) = -\omega \times [Distance(q_t) + \tfrac{1}{n} \times h(s_t, \phi^t_{guiding})]$$

where function h is the so called *domain knowledge function* and is a sort of estimate of needed steps number to satisfy the LTL_f formula from a certain state s. The purpose now is not to dig into the formula and explain how to choose ω and n or what $Distance(q_t)$ function means, there's the original paper for this. The important thing to understand is that if the agent choose a "good" action (from s to s'), $\phi(s)$ increases, PBRS function $F(s, s')$ is positive and indeed we add a positive additional reward.
They evaluated their method on two different variations of the flag collection domain: with *multiple flags* and with an aspect of *uncertainty*. In the first scenario the agent should collect all the flags and go to the goal position in the minimum number of steps. In the second one, there are two flags involved and at each experiment one of the two doors can be closed and the other is open. First of all, they showed how convergence to the optimal policy, with their method, was actually faster than RL algorithms with no domain knowledge involved (see the plots in the original paper [9]). In addition, they've pointed out the flexibility of their LTL_f-based method. In which sense? In the multiple flags version, since there are more than one optimal ordering of collecting them, it provides the flexibility for the user to guide the agent to any optimal ordering he choose. In the case where we don't know which door is open, and then we don't know which is the optimal flags ordering a priori, their method gives the flexibility to not bias the agent to collect one flag before the other one (without pushing a certain ordering). These are in my opinion incredible results and a good starting point to appreciate the integration power of *learning* and *reasoning*.

4 Restraining Bolts

The term comes from Star Wars. A restraining bolt is a device that restricts droid's actions to a set of desired behaviours. In our case the droid is a RL agent and the restraining bolt a logical specification of traces that are considered desirable.
How this logic formulas are used in practice? As in all the methods discussed in this report, temporal specifications, to be operatively employed and actually help RL algorithms, need to be transformed in DFAs (more suitable tools for computations and tracking satisfaction stage of formulas). We know from automata theory that it can be done: from formula φ to DFA $\mathcal{A}_\varphi$, where $\mathcal{A}_\varphi$ accepts a finite trace π if and only if π satisfies φ. Very often we take this result for granted but we have to acknowledge its fundamental importance. We can state that DFAs and *automata* in general (like *transducers* in Section 5) constitute important ingredients in *knowledge* integration with *learning* methodologies.
The paper *Foundations of Restraining Bolts: Reinforcement Learning with LTL$_f$/LDL$_f$ Restraining Specifications* [10], presented to ICAPS 2019, it introduces a novel problem in AI and, in my opinion, paves the way to countless applications. The authors showed that an agent can learn while shaping its goals to suitably conform to restraining specifications imposed by an *authority*. This is extremely powerful because we can control robots behaviour by "simply" specifying some constraints. How? Through temporal logic (LTL$_f$/LDL$_f$). As already mentioned in the introduction, Restraining Bolts also fit in that family of methods which aim to model MDPs (see Section 2) with non-Markovian rewards $R : (S \times A)^* \rightarrow \mathbb{R}$. The design of a reward function is inherently difficult, and its choice significantly impacts policy solutions. Now, consider the added complexity when dealing with a non-Markovian reward function. We need help from KR to define this type of functions and researchers from the field started to combine agent's MDP with temporal logic formulas associated with a reward. From this scenario Restraining Bolts (RBs) are born: defined as $\langle \mathcal{L}, \{\varphi_i, r_i\}_{i=1}^m \rangle$, where $\mathcal{L}$ is the set of possible fluents configurations and $\{\varphi_i, r_i\}_{i=1}^m$ is a set of *restraining specifications* with φ_i a LTL$_f$/LDL$_f$ formula and r_i the reward associated with it.
The goal has always been to derive an equivalent MDP (with logic specifications included) from the original one, in order to being able to use the classical RL methods (*Q-learning* and *SARSA* for example) to solve it. In [11] (presented at AAAI 2018), it was shown that, by having an agent's MDP $M_{ag} = \langle S_{ag}, A_{ag}, Tr_{ag}, R_{ag} \rangle$, a restraining bolt $RB = \langle \mathcal{L}, \{\varphi_i, r_i\}_{i=1}^m \rangle$ and a mapping between the representation of the states S_{ag} and the *fluents* $\mathcal{L}$ used to specify the formulas, is possible to define an equivalent MDP and use *planning* to find a policy. This was the theoretical result from which the authors of [10] based their work. By omitting all the intermediate steps (that can be viewed in the original paper), they ended up with a theorem stating that: a RL problem with LTL$_f$/LDL$_f$ *restraining specifications* $\mathcal{M}_{ag}^{rb} = \langle \mathcal{M}_{ag}, RB \rangle$ can be

reduced to a RL problem over an equivalent MDP $\mathcal{M}^q_{ag}$ and use *Q-learning/SARSA* to find the optimal policy. This new MDP is attained by ignoring transition Tr_{ag} and reward R_{ag} functions, and by augmenting the state representation S_{ag}: $\mathcal{M}^q_{ag} = \langle S \times Q_1 \times ... \times Q_m, A\rangle$, where Q_i are the encodings relative to the current state of each DFA (each φ_i is transformed to a DFA$_i$). Basically, we can implicitly learn non-Markovian rewards by learning Markovian rewards of the transformed MDP $\mathcal{M}^q_{ag}$ (modifying only the state space adding automata informations). To ensure it works in practice, we need to feed the rewards r_i[1] at suitable times and to allow the agent to keep track of DFA$_i$ satisfaction stage during training phase (we need a function which extracts *fluents* from the environment). At each time step the total reward is $R + \sum_{i=1}^{m} r_i$, where R is the rewards coming from the original $\mathcal{M}_{ag}$ and m the number of restraining specifications. All of this can be achieved without the mapping between S^q_{ag} and $\mathcal{L}$ (like it's instead needed in [11]).
For this reason we have a completely decoupled set of features: one for the *KR-based* monitor and one fort the *RL-based* agent. In Figure 4 can be seen the general schema. The two components extract different types of features from the same environment

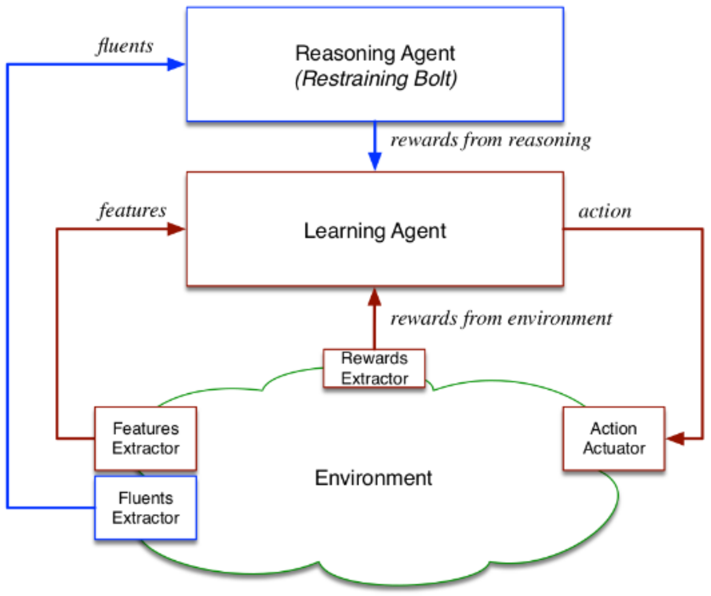


Figure 4: Restraining Bolts schema.

using separate *sensors*, each focusing on a specific part of the world they intend to capture. They ignore each other and they can actually be not aware of each other's existence. Using *Breakout* environment as an example, the *RL-based* sensor may log if a single brick has been destroyed or the position of the ball, while the *KR-based* sensor may measure if an entirely column has been broken. Learning agent gets

[1]additional reward related to DFA$_i$ completion. To speed up learning, it's applied to r_i a kind of reward shaping which exploits the DFA$_i$ structure. It this way we anticipate part of the reward coming from temporal specifications without waiting for the formulas to become true.

the rewards not only from the environment but also an additional reward from the *reasoning* module (as already explained).
This capability of the two parts of being decoupled makes a difference in actual implementations and in possible real-world applications. It promotes a separation of concerns facilitating the design and providing modularity and reuse of representations: we can have different intelligent robots designed by different tech companies and have a single RB designed by the authority who wants to impose some behavioural limitations to them. Or the other way around, we can have various types of RBs all applied to a single agent. As already mentioned in Section 1 this method can be considered part of the world of *safe AI*, but it must be said that doesn't guarantee the hard satisfaction of constraints imposed by the RBs. We need something different!

4.1 Restraining Bolts as *Shields*

Preemptive shields are automata which output, at each timestep, the set of actions that they considers *safe*. In recent years many are the papers which tries to develop *safe RL* algorithms. One popular approach, used in [12], is precisely to use *shields* to constraint agents by allowing them to choose actions only from a subset of the available ones (*action masking*).
In this context, a recently published paper was introduced at the ICAPS conference that took place in Prague this year, named *Preemptive Restraining Bolts* (PRBs) [13]. The authors proposed an approach to implement *non-Markovian action masking*. A Preemptive Restraining Bolt consists of a set of Pure Past Linear-time Temporal Logic (PPLTL) formulas, one per action (see Section 1 for syntax details). They used this type of logic because they claimed that "*many specifications are easier and more natural to express when referring to the past*". How do these PPLTL specifications allow action masking? While the agent acts, at any timestep an action is allowed if and only if its corresponding PPLTL formula is satisfied given the current history.
PBRs owe their name to restraining bolts [10] because like them, PBRs can use a different set of *fluents* from that available to the agents, promoting a separation of concerns between *safety* and *policy optimization*. Although RBs and PRBs are conceptually similar, because they aim to constraint somewhat RL agent's behaviour, they aren't actually. An agent with a RB is free to choose any action in a given state s of the environment, while on the other hand, a PRB by design restricts the set of actions available to an agent in state s to those which are permitted (the *safe* ones) by the restraining specification. While in certain scenarios we can achieve the same result both with RBs and with PRBs, in others, where it's not affordable to choose certain *unsafe* actions (think about autonomous driving and possible fatal choices), the only way is to adopt solutions such as PRBs.
In additon, one notable result of *action masking* to highlight is its capability of speeding

up the optimal policy convergence process. In Figure 5 is shown how the agent trained with PRB converges faster than the one trained with the RB. This makes sense because an agent which uses *preemptive* RBs doesn't explore unnecessary actions, leading to an overall computational efficiency.

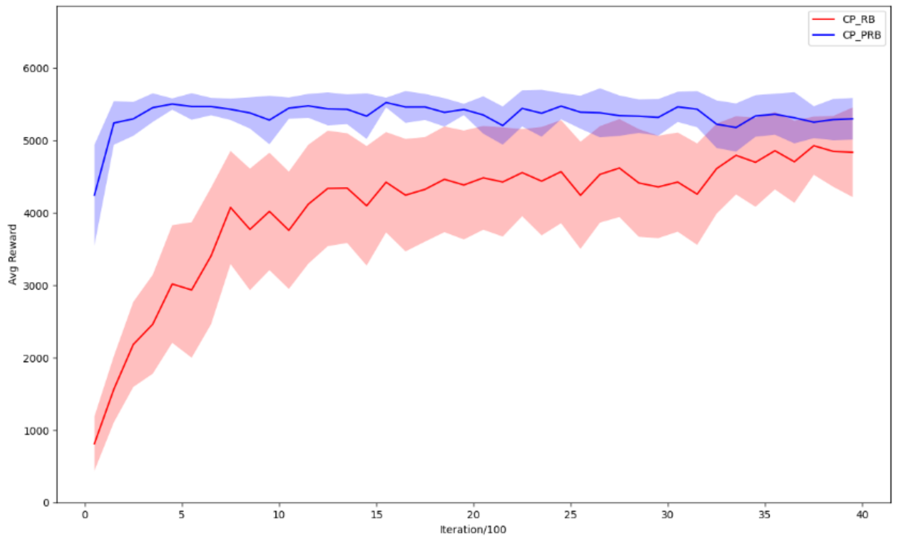


Figure 5: Average reward every 100 iterations. Red line refers to RB agent and the blue one to the PRB one. The exeriments were conducted in the "Cocktail Party" environment, which is part of the RoboCup challenge.

4.2 Imitation Learning

Another way of dealing with the difficulty of expressing the reward function is through *Imitation Learning* (IL). The *learner* agent learns the *expert* agent's behaviour and obtains a reward function from a set of execution traces generated by the latter.
Imitation Learning over Heterogeneous Agents with Restraining Bolts paper presented at ICAPS 2020 [14], shows how this concept can be easily extended to restraining bolts [10]. Indeed, the authors wanted to address the problem of transferring a RB formula φ from an expert to a learner agent. This can be useful either in situations where the restraining behaviours we want to impose are difficult to express in $\text{LTL}_f/\text{LDL}_f$ formulas or in situations when we want to "imitate" an agent's behaviour which knows how to accomplish the particular task but cannot describe it (in other words, it cannot provide its RB). Since the formula φ relative to the RB can be transformed in a DFA, the difficulty of the problem lies in the extraction the *expert* agent's DFA. How can we proceed?
Operatively, as *expert* executes its optimal policy, it produces some DFA traces (remember that it has a RB embodied). This traces can be positive or negative, and are correctly labeled by the agent thanks to its state representation. As a result we have a set $\mathcal{T}$ of (positive and negative) traces which explain which traces are accepted by the DFA

we want to learn and which are not. Since the DFA under consideration is unknown even to the expert, the best we can hope for is to come up with an approximated DFA that accepts all positive and no negative traces according to $\mathcal{T}$. This is the minimum requirement we look for. There are several approaches to extract a DFA from a set $\mathcal{T}$ of labelled traces, but in the paper it's been selected the L^* algorithm [15] (mainly for its simplicity and not for other particular reasons).
Once the approximated DFA (which models the behaviour and the non-markovian reward function of the *expert* agent) has been computed thanks to L^*, it can be immediately plugged into the *learner* agent and used as restraining specification. The power of this method resides in the capability of being applied to *heterogeneous* agents, i.e. agents which use different representations of the environment's actions and states. In fact, in the conducted experiments, I employed two different variants of the *Breakout* game respectively as *expert* and *learner.* The two variants have different state representations, but what really makes the difference is the action space. In the first variant (*expert*), to make the game and the policy learning as easy as possible, the agent can only *fire* to destroy the bricks (no ball involved). Instead, in the second variant, the *learner* learns a policy in a more complex game where there's the ball and the paddle can move *left* and *right.* More informations about the effectiveness of the approach can be found in Section 6.

5 Monitoring Rewards

Temporal Logic Monitoring Rewards via Transducers [16] is a paper published at *KR2020* conference again by Sapienza researchers. Their work originates from the difficulty of rewarding an agent for its behaviour over time, i.e. the difficulty to model non-Markovian reward functions (as it happens with restraining bolts [10]). Their specifications can be cumbersome and error-prone and this *reward hacking* process quite tedious and difficult to carry out.
Imagine the simple *mail delivery* problem, where there's a mailman robot which has to deliver the mail to a certain position and it can accidentally fall into the lake. Mail cannot be delivered anymore if it gets wet. For this reason, the problem under consideration can be framed as an *approach-avoid* task, where the agent has to achieve a certain goal while maintaining a certain condition true (in this case the mail which must remain dry). In this scenario, rewards are wrongly given if only based on the current state (Markov property). We cannot give to the agent a positive reward if it reaches the goal with wet mail! We need to find a way to "remember" if the agent fall into the lake in some previous steps or not.
The authors devised a way of specifying not-Markovian rewards using LTL$_f$/LDL$_f$ formulas which associates reward not to simply the satisfaction of the formula, but to the four classical *monitoring conditions* [17]: the formula can be temporarily true,

temporarily false, permanently true or permanently false. They proved that, through the use of *transducers* (see Section 2), these four conditions can be monitored at no additional cost with respect to satisfaction only. With the idea of extending temporal

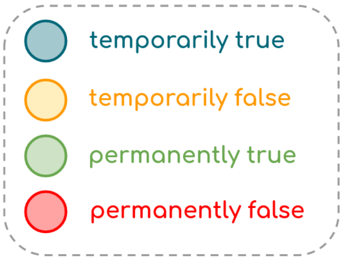


Figure 6: Monitoring conditions legend. Transducers in Figure 8 and 9 are colored following it.

logic reward specifications, they introduced *monitoring rewards*. A monitoring reward is a 5-tuple $\langle \varphi, r, c, s, f \rangle$ where φ is the LTL$_f$/LDL$_f$ formula, r (reward) is the value the agent receives when φ is temporarily true, c (cost) when is temporarily false, s (success) when is permanently true and f (failure) when is permanently false. The authors presented how to realize *monitoring rewards* via transducers, in particular via *reward transducers*, which are essentially transducers that output rewards. They also demonstrated how this type of automata are better than *acceptors* for this kind of purpose in terms, for example, of "combination" capabilities. In fact, the four transducers (relative to the four monitoring conditions) can be "easily" combined through the *sum* operation in a final reward transducer which implements the original *monitoring reward*. The scenario is the one depicted in Figure 7, where, during learning, each environment observation is passed to a *monitor* which interprets the observation in *fluents* terms, updates our reward transducer based on this interpretation and finally outputs a reward. It's indeed very important to have in the actual implementation a function which maps environment's observation to transducer traces. Instead of

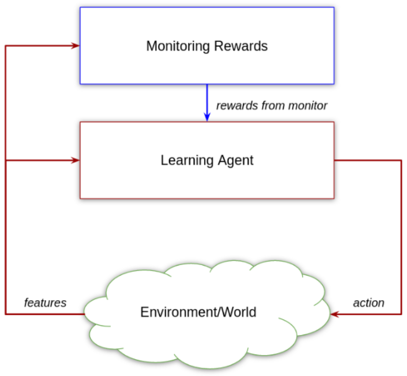


Figure 7: Block structure of the learning process of RL agent with *monitoring rewards*.

doing experiments regarding the implementation of this method, I decided to show

how to build these reward transducers according to two example case scenarios.

Mail Delivery. As already said, this is an *approach-avoid* task that can be expressed by the formula $\neg failure \mathcal{U} goal$. In this family of problems can also be inserted the *Cliff Walking* environment where at the bottom of the grid world, instead of the *lake*, there's a *cliff*. Stepping into this type of region incurs a reward of -100 and makes the simulation to fail. The reward function can be captured by the monitoring specification $\langle \neg failure\ \mathcal{U}\ goal, 0, -1, +1, -100 \rangle$ and the resulted transducer is represented in Figure 8. As long as we neither reached the goal nor failed we received the *cost* -1; as soon as we fail we receive -100 (even if the agent reach the final goal); as soon as we reach the goal and we never failed, we gain a $+1$ *success* reward. Notice that the formula can never be temporarily true.

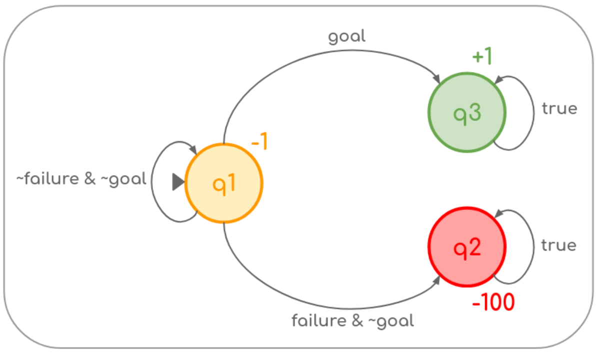


Figure 8: Reward transducer of a general *approach-avoid* task.

Taxi domain. In the taxi domain[2] the goal is to *pick-up* the passenger at one location (4 in total) and then *drop*-him-*off* in another one (a *sequential* task). The formula to model the sequential task is $\diamond(pick\_up \wedge \diamond drop\_off)$. There's also a negative reward (*cost*) of -10 for illegal *pick-up* (if the taxi wants to pick-up in a location where there's no person) and *drop-off* (if the person is not in the taxi and the agent want to perform it) actions. The bad action penalty is modeled by another temporal specification, $\diamond bad\_action$, meaning that we have two reward transducers in total. When there are multiple transducers, at the end of each episode, the total reward is simply the sum of the reward of each transducer and once anyone of them enters an infinite loop, the total reward should be returned. The two monitoring rewards for the Taxi domain are $\langle \diamond(pick\_up \wedge \diamond drop\_off), 0, -1, +20, 0 \rangle$ (Figure 9a) and $\langle \diamond bad\_action, -10, 0, 0, 0 \rangle$ (Figure 9b). Since *pick_up* stands for "pick up the passenger" and *drop_off* for "drop off the passenger to the right location", the transducer in Figure 9a properly expresses the order of occurence of the two *fluents* in order to reach the final goal and receive $+20$ as reward. In addition, given the absence of infinite loops in the transducer of Figure 9b, within an episode, the agent can be penalized as many times as the number of times he performs illegal actions. Obviously, other RL environments

[2]https://gymnasium.farama.org/environments/toy_text/taxi/

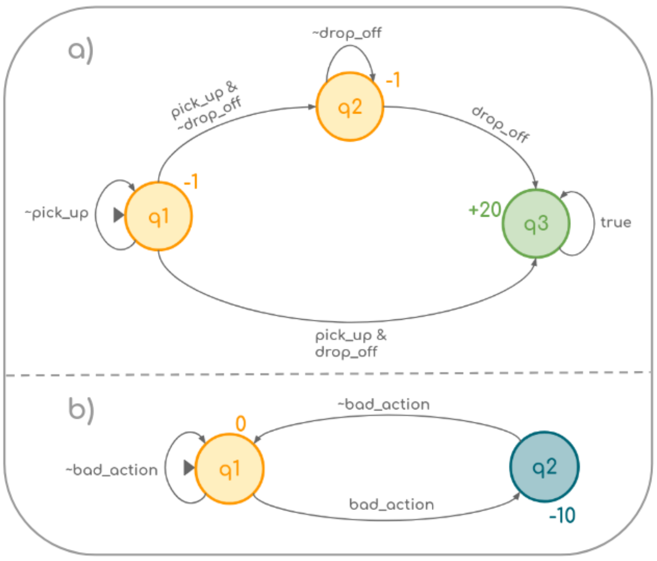


Figure 9: The two reward transducers of the taxi domain.

that have sequential tasks can use monitoring rewards of this nature. For example, the *Breakout* environment where we want to break the columns in a certain order. If we have 3 columns, we want to destroy the bricks columns from left to right and c_i is the *fluent* which is true if "the i-th column has been correctly destroyed", we end up with $\langle \diamond(c_1 \wedge \diamond(c_2 \wedge \diamond c_3)), 0, -1, +20, 0\rangle$ and $\langle \diamond bad\_action\ , -10, 0, 0, 0\rangle$ as monitoring rewards. The *bad_action* in this case is when we break a column not following the pre-fixed order.

6 Experiments

The literature about the union between KR and RL is fascinating and the research papers highly inspiring, but do they really work in practice? I checked it personally by running some experiments, in particular regarding *Restraining Bolts*: both in their standard formulation [10] and in an Imitation Learning (IL) setting [14]. The implementated code can be viewed at the following GitHub repository and all the quantitative plots at this WandB workspace.

6.1 Implementation

The code for running the experiments wasn't implemented all from scratch. I started from the official repository of [14]. My primary focus wasn't a re-implementation of the method, but the execution of the official code to see the effectiveness of it under different scenarios and circumstances (we'll discuss them in Section 6.2). I essentially adapted the official code to my needs, which include the execution of all the experiments in the *Breakout* environment and the possibility to run restraining bolts

with LTL$_f$/LDL$_f$ formulas either already specified, or to be learnt (IL). *Breakout* Atari game, thanks to its many variants (in terms of action and state space), was perfect for this testing scenario.
We used Python as programming language and many libraries and tools to achieve our goal. They are worthy to be mentioned *gym-breakout-pygame* and *temprl* repositories (both implemented by WhiteMech research group from Sapienza University). The former simply contains the environment implementation and the latter permits the integration of temporal goal specifications with the RL environment.
As Reinforcement Learning algorithms we used tabular Temporal Difference (TD) methods (see Section 2 for more details). In particular *Q-learning* and *Sarsa* algorithms. We employed in all the experiments the λ version of them: *Q(λ)-learning* and *Sarsa(λ)*. I didn't spend much time in finetuning hyperparameters since it wasn't the purpose of the research, but, after few trials, it was set $\gamma = 0.99$, $\alpha = 0.1$ and $\lambda = 0.99$.
The goal of Breakout game is to destroy all the bricks. Can we guide the agent to destroy them in a certain order? This is exactly what RBs can do: we can employ a temporal specification where we define in which order we should break the bricks. For example, if we want to break the column on the left first and the column on the right for last, the LDL$_f$ logical formula should be:

$$\varphi = <(\neg c_1 \wedge \neg c_2 \wedge \neg c_3)^*; c_1; (\neg c_1 \wedge \neg c_2 \wedge \neg c_3)^*; c_2; (\neg c_1 \wedge \neg c_2 \wedge \neg c_3)^*; c_3> tt$$

where c_i is a *fluent* that means that the i^{th} column of bricks has just been broken. The correspondent RB is $\{(\varphi, 1000)\}$: expressing that the RL agent will receive an additional reward of 1000 when φ will be satisfied. As already explained in Section 4, this particular reward is simply added to the "normal" ones coming from the environment, that in our case are -0.01 for each step taken and $+5$ for each brick broken.

6.2 Results

The methods have been tested on different scenarios: different bricks configurations (3×3, 4×4 and 8×3), different actions (*move* the paddle and/or *fire*), different orders of breaking bricks (left-to-right *sx2dx* or right-to-left *dx2sx*) and different TD(λ) algorithms (*Q-learning* and *Sarsa*). The number of training iterations were set due to the inherent difficulty of each specific scenario. It can be anticipated that in general *Sarsa* behaves better than *Q-learning*, and, as one might have imagined, the "order of bricks breaking" (*sx2dx* or *dx2sx*) expressed by RB specification didn't influence in any way the convergence of the learning algorithm and in general the agent's behaviour.
In Figure 10 we can see the first set of results of Restraining Bolts applied to a 3×3 bricks configuration in the Breakout game. In this simple scenario, there's no much difference between the agent that can only move the paddle (*BALL*) and the agent that can both move the paddle and fire towards the bricks (*FIRE+BALL*).

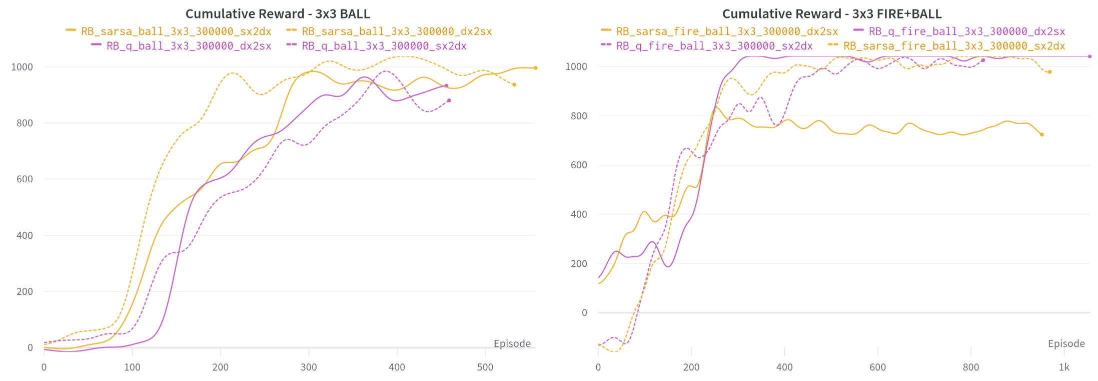


Figure 10: Cumulative Reward plots of RBs applied to Breakout game in 3×3 bricks configuration.

We can slightly start to notice how *Sarsa* performs a little bit better than *Q-learning*. This can be noticed by the faster convergence to the optimal policy of yellow plots (in our case the optimal policy is reached when the total reward is near 1000, since is the additional reward the RB gives to the RL agent when the logical specification is satisfied).
We also run the same experiments on a 4×4 Breakout configuration (see Figure 11). Here things are more complicated and it can be easily recognized how the additional agent's ability to *fire* really helps in winning the game according to RB specification. This can be observed in the number of episodes the agent employed to converge when it also uses *fire* compared to when it only moves the paddle: $\sim 2.5k$ episodes against $\sim 4k$ episodes. Moreover, is clear that in some cases *Q-learning* algorithm (purple lines) doesn't even reach the optimal policy. For all subsequent experiments, since it proved to be better, only the *Sarsa* algorithm will be used.

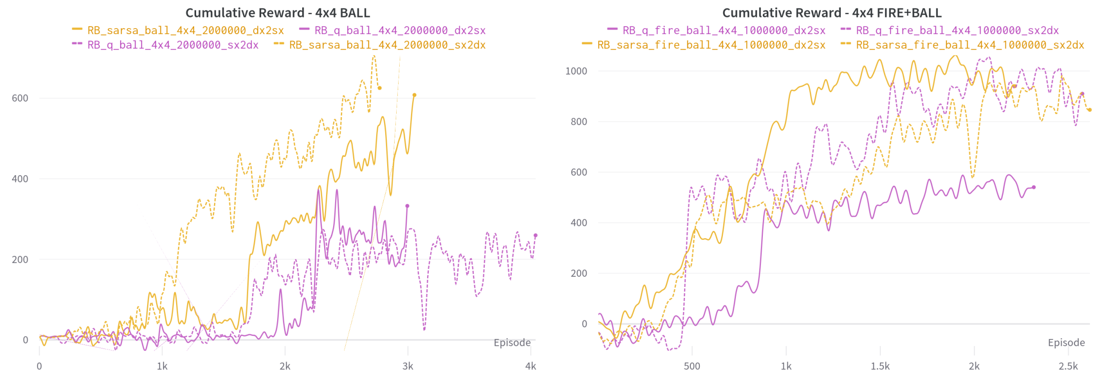


Figure 11: Cumulative Reward plots of RBs applied to Breakout game in 4×4 bricks configuration.

The other experimented scenario was the 8×3 bricks configuration. In this case, being bricks rows more dense and the first row closer to the paddle than previous scenarios, the paddle response time is shorter (i.e. the paddle has less time to figure out what to do because the ball returns immediately). For this reason, the agent which only can move the paddle behaves better than the agent which can also *fire* to the bricks. That's because the *FIRE+BALL* agent wastes time shooting fire to destroy the bricks as quickly as possible, getting distracted, however, by the ball that comes back quickly. The results can be seen in Figure 12.

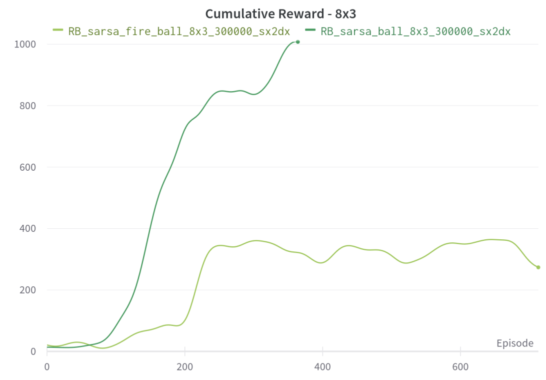


Figure 12: Cumulative Reward plots of RBs applied to Breakout game in 8×3 bricks configuration with *Sarsa* algorithm and *left-to-right* as order of bricks breaking.

Finally, the experiments about Imitation Learning [14] were run. An agent (*learner*) had to imitate the *expert*'s behaviour in order to learn its Restraining Bolt that is not explicitly defined. This is done by observing *expert*'s execution *traces* (i.e. the sequence of states and transitions taken by the DFA correspondent to the RB during training phase). In practice, this process was possible thanks to the *inferrer* library, an automata learning library (made by the WhiteMech research group). For not over complicating this imitation process, the *expert* agent was as simple as possible: no ball involved and it could only *fire*. In Figure 13 it's possible to see how fast it converges to the optimal policy (green plot).

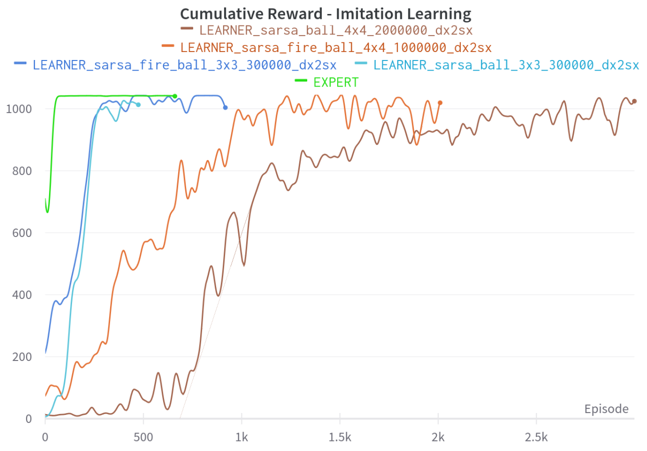


Figure 13: Cumulative Reward plots of RBs applied to Breakout game in an Imitation Learning setting. The *learners* were tested in both 3×3 and 4×4 bricks configuration with *Sarsa* algorithm and *right-to-left* as order of bricks breaking.

All the *learners* learnt to win the game according to RBs: both in 3×3 (blue and

light-blue) and 4×4 (orange and brown) bricks configuration.
Each experiment above mentioned and commented has a policy execution video that can be qualitatively appreciated in the *experiments* folder of the project repository.

7 Conclusions

Reinforcement Learning really needs an help from Knowledge Representation. Many AI researchers realized it and are trying to combine the two worlds, so distant yet complementing each other. Different methods were shown and deeply showcased to prove their practical relevance in real-world applications. Imagine when robots will be more present in our daily lives: we need tools to be able to control them, to limit their behaviours to certain rules and to make them safe without hurting people. One possible way is through the integration of KR techniques. Developing *safe* and *explainable* AI systems is one of the biggest challenges for us which embody the next generation of AI researchers. Deep learning cannot be the answer to everything: we have entered a new AI era, that of the union of *learning* and *reasoning*, or, as I prefer, the *yin and yang of future AI.*

References

[1] Adnan Darwiche. Human-level intelligence or animal-like abilities?, 2017.

[2] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. The computational limits of deep learning, 2022.

[3] Gary Marcus. The next decade in ai: Four steps towards robust artificial intelligence, 2020.

[4] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. IJCAI '13, page 854–860. AAAI Press, 2013.

[5] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[6] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. Pure-past linear temporal and dynamic logic on finite traces. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4959–4965. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Survey track.

[7] Alonzo Church. Edward f. moore. gedanken-experiments on sequential machines. automata studies, edited by c. e. shannon and j. mccarthy, annals of mathematics studies no. 34, litho-printed, princeton university press, princeton1956, pp. 129–153. *The Journal of Symbolic Logic*, 23(1):60–60, 1958.

[8] George H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.

[9] Mahmoud Elbarbari, Kyriakos Efthymiadis, Bram Vanderborght, and Ann Nowe. Ltlf-based reward shaping for reinforcement learning. In *Proceedings of the Adaptive and Learning Agents Workshop 2021 (ALA2021) at AAMAS*, April 2021. Adaptive and Learning Agents Workshop 2021 : at AAMAS , ALA2021 ; Conference date: 03-05-2021 Through 04-05-2021.

[10] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1):128–136, May 2021.

[11] Ronen Brafman, Giuseppe De Giacomo, and Fabio Patrizi. Ltlf/ldlf non-markovian rewards. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[12] Mohammed Alshiekh, Roderick Bloem, Ruediger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding, 2017.

[13] Giovanni Varricchione, Natasha Alechina, Mehdi Dastani, Giuseppe De Giacomo, Brian Logan, and Giuseppe Perelli. Preemptive restraining bolts. In *PRL Workshop Series – Bridging the Gap Between AI Planning and Reinforcement Learning*, 2023.

[14] Giuseppe De Giacomo, Marco Favorito, Luca Iocchi, and Fabio Patrizi. Imitation learning over heterogeneous agents with restraining bolts. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1):517–521, Jun. 2020.

[15] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[16] Giuseppe De Giacomo, Marco Favorito, Luca Iocchi, Fabio Patrizi, and Alessandro Ronca. Temporal Logic Monitoring Rewards via Transducers. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 860–870, 9 2020.

[17] Giuseppe De Giacomo, Riccardo Masellis, Marco Grasso, Fabrizio Maggi, and Marco Montali. Monitoring business metaconstraints based on ltl and ldl for finite traces. 09 2014.