Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf restraining specifications

Giuseppe De Giacomo and **Luca Iocchi** and **Marco Favorito** and **Fabio Patrizi**
DIAG - Università di Roma "La Sapienza", Italy
{*lastname*}@diag.uniroma1.it

Abstract

In this work we investigate on the concept of "*restraining bolt*", envisioned in Science Fiction. Specifically we introduce a novel problem in AI. We have two distinct sets of features extracted from the world, one by the agent and one by the authority imposing restraining specifications (the "restraining bolt"). The two sets are apparently unrelated since of interest to independent parties, however they both account for (aspects of) the same world. We consider the case in which the agent is a reinforcement learning agent on the first set of features, while the restraining bolt is specified logically using linear time logic on finite traces LTL_f/LDL_f over the second set of features. We show formally, and illustrate with examples, that, under general circumstances, the agent can learn while shaping its goals to suitably conform (as much as possible) to the restraining bolt specifications.

Introduction

This work starts a scientific investigation on the concept of "*restraining bolt*", as envisioned in Science Fiction. A restraining bolt is a "device that restricts a droid's [agent's] actions when connected to its systems. Droid owners install restraining bolts to limit actions to a set of desired behaviors."[1] The concept of restraining bolt introduces a new problem in AI. We have two distinct representations of the world, one by the agent and one by the authority imposing restraining specifications, i.e., the bolt. Such representations are apparently unrelated as developed by independent parties, but both model (aspects of) the same world. We want the agent to conform (as much as possible) to the restraining specifications, even if these are not expressed in terms of the agent's world representation.

Studying this problem from a classical Knowledge Representation perspective (Reiter 2001) would require to establish some sort of "glue" between the representation by the agent and that by the restraining bolt. Instead, we bypass dealing with such a "glue" by studying this problem in the context of Reinforcement Learning (RL) (Puterman 1994; Sutton and Barto 1998), which is currently of great interest to develop components with forms of decision making, possibly coupled with deep learning techniques (Mnih et al. 2015; Silver et al. 2017).

Specifically, we consider an agent and a restraining bolt of different nature. The *agent* is a reinforcement learning agent whose "model" of the world is a hidden, factorized, Markov Decision Processes (MDP) over a certain set of world features. That is, the state is factorized in a set of features observable to the agent, while transition function and reward function are hidden. The *restraining bolt* consists in a logical specification of traces that are considered desirable. The world features that are used represent states in these traces are disjoint from those used by the agent. More concretely such specifications are expressed in full-fledged temporal logics over finite traces, LTL_f and its extension LDL_f (De Giacomo and Vardi 2013; De Giacomo and Rubin 2018; Brafman, De Giacomo, and Patrizi 2018). Notice that the restraining bolt does not have an explicit model of the dynamics of the world, nor of the agent. Still it can assess if a given trace generated by the execution of the agent in the world is desiderabile, and give additional rewards when it does.

The connection between the agent and the restraining bolt is loose: the bolt provides additional reward to the agent and only needs to know the order of magnitude of the original rewards of the agent to suitably fix a scaling factor[2] for its own additional rewards. In addition, it provides to the agent additional features to allow the agent to know at what stage of the satisfaction of temporal formulas the world is so that the agent can choose its policy accordingly. Without them, the agent would not be able to act differently at different stages to get the rewards according to the temporal specifications.

The main result of this paper is that, in spite of the loose connection between the two models, under general circumstances, the *agent can learn to act so as to conform as much as possible to the* LTL_f/LDL_f *specifications*. Observe that we deal with two separate representations (i.e., two distinct sets of features), one for the agent and one for the bolt, which are apparently unrelated, but in reality, correlated by the world itself, cf., (Brooks 1991). The crucial point is that, in order to perform RL effectively in presence of a restraining bolt *such a correlation does not need to be formalized*.

For example, consider a service robot serving drinks and

[1] `https://www.starwars.com/databank/restraining-bolt`

[2] Note that finding the right scaling factor is an importan issue in RL (Simsek and Barto 2006), but out of the scope of the paper.

snacks at a party. The robot knows the locations where it can grasp drink and snack items and the locations of people that can receive such items. The robot can execute actions to move in the environment, to grasp objects and to deliver them to people. With rewards associated to effective deliver, the robot can learn how to serve something to a specific person. Now, suppose we want to impose the following restraining bolt specification: *serve exactly one drink and one snack to every person, and do not serve alcoholic drinks to minors*. To express this specification (e.g., as an LTL_f/LDL_f formula), a representation of the status of each person (i.e., identity, age and received items) is needed, even though these features are *not* available to the learning agent (but only to the restraining bolt).

Notice that the presence of LTL_f/LDL_f specification makes the whole system formed by the agent and the restraining bolt non-Markovian. Recently, interest in non-Markovian Reward Decision Processes NMRDPs (Bacchus, Boutilier, and Grove 1996; Whitehead and Lin 1995), and in particular on expressing rewards using linear-time temporal logic has been revived and motivated by the difficulty in rewarding complex behaviors directly on MDPs (Littman 2015; Littman et al. 2017). In this context, the use of linear time logics over finite traces such as LTL_f or its extension LDL_f has been recently advocated (Camacho et al. 2017; Brafman, De Giacomo, and Patrizi 2018; Icarte et al. 2018). Notably, both LTL_f and LDL_f formulas can be transformed into deterministic finite state automata tracking the stage of satisfaction of the formulas (De Giacomo and Vardi 2013). This, in turn, allows for transforming an NMRDP with non-Markovian LTL_f/LDL_f rewards into an equivalent MDP over an extended state space, obtained as the crossproduct of the states of the NMRDP and the states of the automaton. This transformation is of particular interest in RL with temporally specified rewards expressed in LTL_f/LDL_f, since it allows to do RL on an equivalent MDP whose (optimal) policies are also (optimal) policies for the original problem, and viceversa (Brafman, De Giacomo, and Patrizi 2018). This provides the basis for our development here.

In this paper, we set the framework for the problem of restraining bolt in RL context and provide proofs and practical evidence, through various examples, that an RL agent can learn policies that optimize conformance to the LTL_f/LDL_f restraining specifications, without including in the agent's state space representation the features needed to evaluate the LTL_f/LDL_f formula. Our work can also be seen as a contribution to the research providing safety guarantees to AI techniques based on learning. We take up this point in a brief discussion at the end of the paper.

Preliminaries

MDP's. A Markov Decision Process (MDP) $\mathcal{M} = \langle S, A, Tr, R\rangle$ contains a set S of states, which in this paper we consider factored into world features, a set A of actions, a transition function $Tr : S \times A \to Prob(S)$ 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 (a real value) received by the agent when transitioning from state s to state s' by applying action a. A solution to an MDP is a function, called a *policy*, assigning an action to each state, possibly conditioned on past states and actions. The *value* of a policy ρ at state s, denoted $v^\rho(s)$, is the expected sum of (possibly discounted by a factor γ, with $0 \leq \gamma\ 1$) rewards when starting at state s and selecting actions based on ρ.

RL is the task of learning a possibly optimal policy, from an initial state s_0, on an MDP where only S and A are known, while Tr and R are not. Typically, the MDP is assumed to start in an initial state s_0, so policy optimality is evaluated wrt $v^\rho(s_0)$. Every MDP has an *optimal* policy ρ^*. In discounted cumulative settings, there exists an optimal policy that is *Markovian* $\rho : S \to A$, i.e., ρ depends only on the current state, and deterministic (Puterman 1994).

LTL_f/LDL_f. The logic LTL_f is the classical linear time logic LTL (Pnueli 1977) interpreted over finite traces, formed by a finite (instead of infinite, as in LTL) sequence of propositional interpretations(De Giacomo and Vardi 2013). Given a set $\mathcal{P}$ of boolean propositions, here called *fluents* (Reiter 2001), LTL_f formulas φ are defined as follows:

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

where ϕ is a propositional formula over $\mathcal{P}$, $\circ$ is the *next* operator and $\mathcal{U}$ is the *until* operator. We use the standard abbreviations: $\varphi_1 \vee \varphi_2 \doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$; *eventually* as $\Diamond\varphi \doteq true\,\mathcal{U}\,\varphi$; *always* as $\Box\varphi \doteq \neg\Diamond\neg\varphi$; week next $\bullet\varphi \doteq \neg\circ\neg\varphi$ (note that on finite traces $\neg\circ\varphi \not\equiv \circ\neg\varphi$); and $Last \doteq \bullet false$ denoting the end of the trace. LTL_f is as expressive as first-order logic (FO) over finite traces and star-free regular expressions (RE).

LDL_f is a proper extension of LTL_f, which is as expressive as monadic second-order logic (MSO) over finite traces (De Giacomo and Vardi 2013). LDL_f allows for expressing regular expressions over such sequences, hence mixing procedural and declarative specifications, as advocated in some work in Reasoning about Action and Planning (Levesque et al. 1997; Baier et al. 2008). Formally, LDL_f formulas φ are built as follows:

$$\begin{array}{lcl} \varphi & ::= & tt \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\varrho\rangle\varphi \\ \varrho & ::= & \phi \mid \varphi? \mid \varrho_1 + \varrho_2 \mid \varrho_1 ; \varrho_2 \mid \varrho^* \end{array}$$

where: tt stands for logical true; ϕ is a propositional formula over $\mathcal{P}$; ϱ denotes path expressions, i.e., RE over propositional formulas ϕ with the addition of the test construct $\varphi?$ typical of Propositional Dynamic Logic (PDL). We use abbreviations $[\varrho]\varphi \doteq \neg\langle\varrho\rangle\neg\varphi$ as in PDL. Intuitively, $\langle\varrho\rangle\varphi$ states that, from the current step in the trace, there exists an execution satisfying the RE ϱ such that its last step satisfies φ, while $[\varrho]\varphi$ states that, from the current step, all executions satisfying the RE ϱ are such that their last step satisfies φ. Tests are used to insert into the execution path checks for satisfaction of additional LDL_f formulas.

For an LTL_f/LDL_f formula φ, we can construct a deterministic finite state automaton (DFA) (Rabin and Scott 1959) $\mathcal{A}_\varphi$ that tracks satisfaction of φ: $\mathcal{A}_\varphi$ accepts a finite trace π *iff* π satisfies φ.[3]

[3]An analogous transformation to automata applies to several

NMRDP's. A non-Markovian reward decision process (NMRDP) (Bacchus, Boutilier, and Grove 1996) is a tuple $M = \langle S, A, Tr, \bar{R}\rangle$, where S, A and Tr are as in an MDP, but the reward $\bar{R}$ is a real-valued function over finite state-action sequences (referred to as *traces*), i.e., $\bar{R} : (S \times A)^* \to \mathbb{R}$. Given a (possibly infinite) trace $\pi = \langle s_0, a_1, \ldots, s_{n-1}, a_n\rangle$, the *value* of π is: $v(\pi) = \sum_{i=1}^{|\pi|} \gamma^{i-1} \bar{R}(\langle \pi(1), \pi(2), \ldots, \pi(i)\rangle)$, where $0 < \gamma \leq 1$ is the discount factor and $\pi(i)$ denotes the pair (s_{i-1}, a_i). In NMRDP's, policies are also non-Markovian $\bar{\rho} : S^* \to A$. Since every policy induces a distribution over the set of possible infinite traces, we can define the value of a policy $\bar{\rho}$, given an initial state s, as: $v^{\bar{\rho}}(s) = E_{\pi \sim M, \bar{\rho}, s} v(\pi)$. That is, $v^{\bar{\rho}}(s)$ is the expected value of infinite traces, where the distribution over traces is defined by the initial state s, the transition function Tr, and the policy $\bar{\rho}$.

Specifying a non-Markovian reward function explicitly is cumbersome and unintuitive, even if only a finite number of traces are to be rewarded. LTL$_f$/LDL$_f$ provides an intuitive and convenient language rewards (Alberto et al. 2017; Brafman, De Giacomo, and Patrizi 2018). Following, (Brafman, De Giacomo, and Patrizi 2018) we can specify $\bar{R}$ implicitly, using a set of pairs $\{(\varphi_i, r_i)\}_{i=1}^m$, with φ_i an LTL$_f$/LDL$_f$ formula selecting the traces to reward, and r_i the reward assigned to those traces, where the atomic propositions, i.e., the fluents, of φ_i correspond to boolean features, or boolean propositions (e.g., relational value comparison) over the world features, forming the components of the state vector. Intuitively, if the current (partial) trace is $\pi = \langle s_0, a_1, \ldots, s_{n-1}, a_n\rangle$, the agent receives at s_n a reward r if φ_i is satisfied by π. Formally, $\bar{R}(\pi) = r$ if $\pi \models \varphi$ and $\bar{R}(\pi) = 0$, otherwise.

NMRDP with LTL$_f$/LDL$_f$ rewards

Before looking at the restraining bolt problem, we review RL for NMRDP's with LTL$_f$/LDL$_f$ rewards.

In (Brafman, De Giacomo, and Patrizi 2018) it is shown that for any NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$, there exists an MDP $M' = \langle S', A, Tr', R'\rangle$ that is *equivalent* to M in the sense that the states of M can be (injectively) mapped into those of M' in such a way that corresponding (under the mapping) states yield same transition probabilities, and corresponding traces have same rewards (Bacchus, Boutilier, and Grove 1996). Denoting with $\mathcal{A}_{\varphi_i} = \langle 2^{\mathcal{P}}, Q_i, q_{i0}, \delta_i, F_i\rangle$ (notice that $S \subseteq 2^{\mathcal{P}}$ and δ_i is total) the DFA associated with φ_i, the equivalent MDP M' is as follows:

- $S' = Q_1 \times \cdots \times Q_m \times S$;
- $Tr' : S' \times A \times S' \to [0, 1]$ is defined as:

$$Tr'(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \begin{cases} Tr(s, a, s') & \text{if } \forall i : \delta_i(q_i, s') = q'_i \\ 0 & \text{otherwise;} \end{cases}$$

- $R' : S' \times A \times S' \to \mathbb{R}$ is defined as:

$$R'(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \sum_{i : q'_i \in F_i} r_i$$

Observe that the state space of M' is the product of the state spaces of M and $\mathcal{A}_{\varphi_i}$, and that the reward R' is Markovian. In other words, the (stateful) structure of the LTL$_f$/LDL$_f$ formulas φ_i used in the (non-Markovian) reward of M is *compiled* into the states of M'.

Theorem 1 ((Brafman, De Giacomo, and Patrizi 2018)). *The NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$ is equivalent to the MDP $M' = \langle S', A, Tr', R'\rangle$ defined above.*

Actually this theorem can be refined, into a stronger lemma that we will use in the following. A policy ρ for an NMRDP M and a policy ρ' for an equivalent MDP M' are *equivalent* if they *guarantee the same rewards*. Assume M' is constructed as above and let ρ' be a policy for M'. Consider a trace $\pi = \langle s_0, a_1, s_1, \ldots, s_{n-1}, a_n\rangle$ of M and assume it leads to state s_n. Further, let q_i be the state of $\mathcal{A}_{\varphi_i}$ on input π. We define the (non-Markovian) policy $\bar{\rho}$ equivalent to ρ' as $\bar{\rho}(\pi) = \rho'(q_1, \ldots, q_m, s_n)$. Similarly, given a policy ρ for M, by just tracking the state of the DFAs $\mathcal{A}_{\varphi_i}$, it is immediate to define the equivalent policy ρ' for M'. Hence we have:

Lemma 2 ((Brafman, De Giacomo, and Patrizi 2018)). *Given an NMRDP M and an equivalent MDP M', every policy ρ' for M' has an equivalent policy $\bar{\rho}$ for M and viceversa.*[4]

RL for NMRDP with LTL$_f$/LDL$_f$ rewards

We are interested in RL in the setting introduced above: learn a (possibly optimal) policy for an NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$, whose rewards r_i are offered on traces specified by LTL$_f$/LDL$_f$ formulas φ_i and where the LTL$_f$/LDL$_f$ reward formulas $\{(\varphi_i, r_i)\}_{i=1}^m$ and the transitions function Tr is hidden to the learning agent.[5] Formally, given M, with Tr and $\{(\varphi_i, r_i)\}_{i=1}^m$ hidden to the learning agent but sampled during learning, and an initial state $s_0 \in S$, the RL problem over M consists in learning an optimal policy $\bar{\rho}$. Notice that, since NMRDP rewards are based on traces, instead of state-action pairs, typical learning algorithms, such as Q-learning or SARSA (Sutton and Barto 1998), which are based on MDPs, are not applicable. However, by the above results an optimal policy for M can be obtained by learning, instead, an optimal policy for M'. Being M' an MDP, this can be done by typical algorithms such as Q-learning or SARSA. Of course, neither M nor M' are (completely) known to the learning agent, and the transformation is never done explicitly. Rather, during the learning process, the agent assumes that the underlying model has the form of M' instead of that of M.

other formalisms for representing temporal specifications over finite traces, including *Past* LTL, *co-safe* LTL, etc. (Bacchus, Boutilier, and Grove 1996; Thiébaux et al. 2006; Slaney 2005; Gretton 2007; 2014; Lacerda, Parker, and Hawes 2014; 2015). All results presented here apply to those formalisms as well.

[4]A variant of this lemma, talking about optimal policy only as originally presented in (Bacchus, Boutilier, and Grove 1996).

[5]Observe that φ_i is over all world features, since we do not distinguish agent features from restraining bolt features yet.

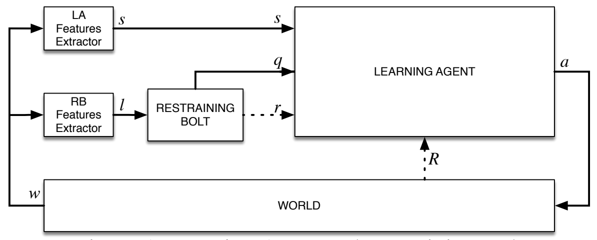

Figure 1: Learning Agent and Restraining Bolt

Theorem 3. *RL for* LTL$_f$/LDL$_f$ *rewards over an NMRDP* $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$*, with* Tr *and* $\{(\varphi_i, r_i)\}_{i=1}^m$ *hidden to the learning agent can be reduced to RL over the MDP* $M' = \langle S', A, Tr', R' \rangle$ *defined above, with* Tr' *and* R' *hidden to the learning agent.*

Note that S' contains encoding of the stage of satisfaction of the formulas φ_i. However since the transition function Tr' is hidden, the agent cannot anticipate the effect of an action before the action is executed.

RL with LTL$_f$/LDL$_f$ restraining specifications

We now focus on the restraining bolt problem, i.e., how to do RL with restraining specifications expressed in LTL$_f$/LDL$_f$.

We are given:

- A **learning agent** modeled by the MDP $M_{ag} = \langle S, A, Tr_{ag}, R_{ag} \rangle$, with transitions Tr_{ag} and rewards R_{ag} hidden, but sampled from the environment.
- A **restraining bolt** $RB = \langle \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$ where:
  - $\mathcal{L} = 2^{\mathcal{F}}$ is the set of possible fluents' configurations (analogously to S denoting the set of configurations of the features available to M_{ag}). Fluents in $\mathcal{F}$ are not among the features that form the states S of the learning agent M_{ag}.
  - $\{(\varphi_i, r_i)\}_{i=1}^m$ is a set of *restraining specifications* with
    - φ_i, an LTL$_f$/LDL$_f$ formula over $\mathcal{F}$. Each φ_i selects sequences of fluents' configurations $\ell_1, \cdots, \ell_n$ ($\ell_k \in \mathcal{L}$) whose relationship with the sequences of states $s_1, \ldots, s_n$ ($s_k \in S$) of M_{ag} is unknown.
    - r_i, the reward associated with φ_i. A reward r_i is assigned to sequences of configurations $\ell_1, \cdots, \ell_n$ satisfying φ_i.

The agent receives rewards based on R_{ag} and the pairs (φ_i, r_i). In fact, often we have to handle tasks of episodic nature. That is, the world can reach a configuration in which no action can change its configuration nor generate new rewards, e.g., a final configuration in a game. In this case we assume that the restraining bolt fluents $\mathcal{F}$ include a special fluent $Done$ that denotes reaching the final configuration. This fluent can be used in LTL$_f$/LDL$_f$ formulas to reward the agent only at the end of the episode. 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.

Notice that while the agent can see the features that the reward R_{ag} depends on, it cannot see those that affect r_i. Both S and $\mathcal{L}$ are features' configurations, in the sense of representing world properties. However, they capture different facets of the world. Let W be the set of *real world states*. A *feature* is a function $f_j : W \to D_j$ that maps world states to another domain D_j, such as reals, enumerations, booleans, etc. The *feature vector* of a world state w_h is the vector $\mathbf{f}(w_h) = \langle f_1(w_h), \ldots, f_d(w_h) \rangle$ of feature values corresponding to w_h. Given a world state w_h, the corresponding *configuration* s_h *of the learning agent* M_{ag} consists in those components of $\mathbf{f}(w_h)$ that produce the agent's state, while the corresponding *configuration of fluents* ℓ_h is formed by the components that assign truth values to the fluents. That is, a subset of the world features describes the agent states s_h and another subset (for simplicity, assumed disjoint from the previous one) is used to evaluate the fluents in ℓ_h. Hence, a sequence $w_1, \ldots, w_n$ of world states defines both a sequence of learning agent states $s_1, \ldots, s_n$ and a sequence of fluent configurations $\ell_1, \ldots, \ell_n$. While R_{ag} depends on the former, each φ_i and r_i depend on the latter. Consequently, by executing a policy and hence by repeatedly choosing actions in A, the agent visits a sequence of world states, collecting for each of them, the sum of the rewards R_{ag} and r_i. The point to resolve is defining on the base of which observations the agent can choose its next actions. Obviously the agent can in principle accumulate all its observations $s_1, \ldots, s_n$ and on the other hand it cannot see the fluents configurations $\ell_1, \ldots, \ell_n$, however we want to equip the agent to some means to establish the stage of satisfaction of the formulas φ_i. Such a notion, as mentioned above, can be captured by considering *the* minimal DFA $\mathcal{A}_{\varphi_i} = \langle 2^{\mathcal{P}}, Q_i, q_{i0}, \delta_i, F_i \rangle$ corresponding to formula φ_i. Notice that such a DFA is unique. Hence we equip the agent with additional observable features $Q_1 \times \ldots \times Q_m$ corresponding to the states Let $\mathcal{A}_{\varphi_i}$. Such features are going to be provided by the restraining bolt.[6] Note that this does not give away fluents configurations $\ell_1, \ldots, \ell_n$ which remain hidden to the agent, see Figure1

Hence, in general, we consider possibly non-Markovian policies of the form

$$\bar{\rho} : (Q_1 \times \ldots \times Q_m \times S)^* \to A$$

and thus define the expected (discounted) cumulative reward of a possibly non-Markovian policy $\bar{\rho}$ as the expected reward of infinite traces starting in the initial state s_0, induced by the policy itself (obtained as the expected sum of the collected rewards R_{ag} and r_i).

Problem definition. (An instance of) the *RL 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 φ_i over $\mathcal{L}$ with associated rewards r_i.

[6]Notice that coming up with the $Q_1, \ldots, Q_n$ and assigning the rewards to some of them, while can perhaps be possible in very simple cases, without a principled and systematic technique as the one presented here it is virtually impossible. Indeed, to express directly LTL$_f$/LDL$_f$ properties in the MDP one may need exponential additional features, assuming a factorized representation, since the corresponding DFA may be doubly exponential in the formula.

A solution to the problem is a (possibly non-Markovian) policy $\bar{\rho} : (Q_1 \times \ldots \times Q_m \times S)^* \to A$ that maximizes the expected cumulative reward.

To devise a solution technique, we assume that the agent actions in A induce a transition distribution over the features and fluents configuration, i.e.:[7]

$$Tr_{ag}^{rb} : S \times \mathcal{L} \times A \to Prob(S \times \mathcal{L}).$$

Such a transition distribution, together with the initial values of the fluents ℓ_0 and of the agent state s_0, allow us to describe a probabilistic transition system accounting for the dynamics of the fluents and agent states. Moreover, when Tr_{ag}^{rb} is projected on S only, i.e., the $\mathcal{L}$ components are marginalized, we get Tr_{ag} of M_{ag}. Obviously, both Tr_{ag}^{rb} and Tr_{ag} are hidden to the learning agent. On the other hand, in response to an agent action a_h performed in the current state w_h (in the state s_h of the agent and the configuration ℓ_h of the fluents), the world changes into w_{h+1} from which s_{h+1} and ℓ_{h+1} are obtained. This is all we need to proceed.

Given $M_{ag}^{rb} = \langle M_{ag}, RB\rangle$ with $M_{ag} = \langle S, A, Tr_{ag}, R_{ag}\rangle$ and $RB = \langle \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$, we define an NMRDP $M_{ag}^n = \langle S \times \mathcal{L}, A, Tr_{ag}^{rb}, \{(\varphi_i, r_i)\}_{i=1}^m \cup \{(\varphi_s, R_{ag}(s,a,s'))\}_{s\in S, a\in A, s'\in S}\rangle$, where:

- states are pairs (s, ℓ) formed by an agent configuration s and a fluents configuration ℓ;
- φ_i are as before;
- $\varphi_s = \Diamond(s \wedge a \wedge \circ(Last \wedge s'))$;
- Tr_{ag}^{rb}, r_i and $R_{ag}(s,a,s')$ are hidden and sampled from the environment.

Formulas φ_i are as before, in particular they are continuously evaluated on the (partial) trace produced so far. Though, they may use the special fluent $Done$ to give the reward associated to the formula at the end of the episode (modulo reward shaping). Formulas $\Diamond(s \wedge a \wedge \circ(Last \wedge s'))$, one per (s,a,s'), which require that both states s and action a are followed by s', are evaluated at the end of the current (partial) trace (recall that $Last$ denotes the last element of the trace, c.f. Preliminaries). In this case, the reward $R_{ag}(s,a,s')$ from M_{ag} associated with (s,a,s') is given.[8]

Observe that policies for M_{ag}^n have the form $(S \times \mathcal{L})^* \to A$ which needs to be restricted to have the form required by our problem M_{ag}^{rb}. A policy $\bar{\rho} : (S \times \mathcal{L})^* \to A$ *has the form* $\bar{\rho} : (Q_1 \times \ldots \times Q_m \times S)^* \to A$ when $\bar{\rho}(\langle s_1, \ell_1\rangle \cdots \langle s_n, \ell_n\rangle) = \bar{\rho}(\langle q_{11}, \ldots, q_{m1}, s_1, \rangle \cdots \langle q_{1n}, \ldots, q_{mn}, s_n, \rangle)$ with $q_{ij} = \delta_j(\ell_1, \ldots, \ell_i, q_{j0})$. In other words, a policy $\bar{\rho} : (S \times \mathcal{L})^* \to A$ has the form $\bar{\rho} : (Q_1 \times \ldots \times Q_m \times S)^* \to A$ when the fluents $\mathcal{L}$ are not directly accessible but are used only the progress the DFAs $\mathcal{A}_{\varphi_i}$ corresponding to formulas φ_i. We can now state the following result.

[7]Notice that this assumption is quite loose, as we can arbitrarily enlarge $\mathcal{L}$ to define Tr_{ag}^{rb}. In the construction below only the fluents in $\mathcal{L}$ that occur in the LTL$_f$/LDL$_f$ formulas play an active role.

[8]Notice that we have as many of such formulas as transitions (s,a,s'), this causes an exponential blow-up being S factorized in features. However, we will get rid of them later.

Lemma 4. *RL with* LTL$_f$/LDL$_f$ *restraining specifications* $M_{ag}^{rb} = \langle M_{ag}, RB\rangle$ *with* $M_{ag} = \langle S, A, Tr_{ag}, R_{ag}\rangle$ *and* $RB = \langle \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$ *can be reduced to RL over the NMRDP* $M_{ag}^n = \langle S \times \mathcal{L}, A, Tr_{ag}^{rb}, \{(\varphi_i, r_i)\}_{i=1}^m \cup \{(\varphi_s, R_{ag}(s,a,s'))\}_{s\in S, a\in A, s'\in S}\rangle$*, by restricting the policy to learn to have the form* $\bar{\rho} : (Q_1 \times \ldots \times Q_m \times S)^* \to A$.

As a second step, we apply the construction of the previous section and obtain a new MDP learning agent. In such construction, because of their triviality, we do not need to keep track of the state of the automata associated with each φ_s, but just offer the reward $R_{ag}(s,a,s')$ associated with (s,a,s'). Instead, we do need to keep track of state of each DFA $\mathcal{A}_{\varphi_i} = \langle 2^{\mathcal{P}}, Q_i, q_{i0}, \delta_i, F_i\rangle$ corresponding to φ_i. Hence, from M_{ag}^n, we obtain an MDP $M'_{ag} = \langle S', A, Tr'_{ag}, R'_{ag}\rangle$ where:

- $S' = Q_1 \times \cdots \times Q_m \times S \times \mathcal{L}$ is the set of states;
- $Tr'_{ag} : S' \times A \times S' \to [0,1]$ is defined as follows:

$$Tr'_{ag}(q_1, \ldots, q_m, s, \ell, a, q'_1, \ldots, q'_m, s', \ell') = \begin{cases} Tr_{ag}(s, \ell, a, s', \ell') & \text{if } \forall i : \delta_i(q_i, \ell') = q'_i \\ 0 & \text{otherwise;} \end{cases}$$

- $R'_{ag} : S' \times A \times S' \to \mathbb{R}$ is defined as:

$$R'_{ag}(q_1, \ldots, q_m, s, \ell, a, q'_1, \ldots, q'_m, s', \ell') = \sum_{i:q'_i \in F_i} r_i + R_{ag}(s,a,s')$$

Observe that, besides the rewards $R_{ag}(s,a,s')$ of the original learning agent, the environment now offers the rewards r_i associated with the formulas φ_i, thus guiding the agent towards the satisfaction of the φ_i (by progressing correctly the corresponding DFAs $\mathcal{A}_{\varphi_i}$).

By Theorem 1, it follows that the NMRDP M_{ag}^n and the MDP M'_{ag} are equivalent. Hence, by Lemma 2, any policy of M_{ag}^n has an equivalent policy (hence guaranteeing the same reward) in M'_{ag}, and viceversa. We can thus learn a policy on M'_{ag} instead of M_{ag}^n. We can thus refine Lemma 4 into the following.

Lemma 5. *RL with* LTL$_f$/LDL$_f$ *restraining specifications* $M_{ag}^{rb} = \langle M_{ag}, RB\rangle$ *with* $M_{ag} = \langle S, A, Tr_{ag}, R_{ag}\rangle$ *and* $RB = \langle \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$ *can be reduced to RL over the MDP* $M'_{ag} = \langle S', A, Tr'_{ag}, R'_{ag}\rangle$*, by restricting the policy to learn to have the form* $Q_1 \times \ldots \times Q_n \times S \to A$.

This Lemma allows for restricting non-Markovian policies $(Q_1 \times \ldots \times Q_n \times S)^* \to A$ to Markovian policy $Q_1 \times \ldots \times Q_n \times S \to A$ without loss of generality.

As a last step, we solve the original RL task on M_{ag}^{rb} by performing RL on a new MDP $M_{ag}^q = \langle Q_1 \times \cdots \times Q_m \times S, A, Tr''_{ag}, R''_{ag}\rangle$, where:

- The transition distribution Tr''_{ag} is the marginalization of Tr'_{ag} wrt $\mathcal{L}$, and is unknown;
- The reward R''_{ag} is defined as:

$$R''_{ag}(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \sum_{i:q'_i \in F_i} r_i + R_{ag}(s,a,s').$$

- The states q_i of the DFAs $\mathcal{A}_{\varphi_i}$ are progressed correctly by the environment.

Thus, we finally obtain our main result.

Theorem 6. *RL with* LTL$_f$/LDL$_f$ *restraining specifications* $M_{ag}^{rb} = \langle M_{ag}, RB\rangle$ *with* $M_{ag} = \langle S, A, Tr_{ag}, R_{ag}\rangle$ *and* $RB = \langle \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$ *can be reduced to RL over the MDP* $M_{ag}^q = \langle Q_1 \times \cdots \times Q_m \times S, A, Tr''_{ag}, R''_{ag}\rangle$ *and optimal policies* ρ_{ag}^{new} *for* M_{ag}^{rb} *can be learned by learning corresponding optimal policies for* M_{ag}^q.

Proof. For brevity we use $\vec{q}$ to denote $q_1, \ldots, q_m$. By Lemma 5 we can focus on RL over the MDP $M'_{ag} = \langle S', A, Tr'_{ag}, R'_{ag}\rangle$ under the restriction that the policy to learn has the form $Q_1 \times \ldots \times Q_n \times S \rightarrow A$.

Notice that from the definitions of R'_{ag} and R''_{ag}, we have that for all $\ell, \ell' \in \mathcal{L}$, $R'_{ag}(\vec{q}, s, \ell, a, \vec{q}', s', \ell') = R''_{ag}(\vec{q}, s, a, \vec{q}', s') = \sum_{i:q'_i \in F_i} r_i + R_{ag}(s, a, s')$. The crux of the proof is to show that for any optimal policy ρ the values $v^\rho(\vec{q}, s, \ell)$ of the state value function for $\mathcal{M}'_{ag}$ do not depend on ℓ. That is, it is necessary that $\forall \ell_1, \ell_2 . v^\rho(q_1, \ldots, q_m, s, \ell_1) = v^\rho(q_1, \ldots, q_m, s, \ell_2)$.

To see this, let $Tr'_{ag}(s, a, s') = P(s'|s, a)$, then the Bellman equation in our case is: $v^\rho(\vec{q}, s, \ell) = \sum_{\vec{q}', s', \ell'} P(\vec{q}', s', \ell'|\vec{q}, s, \ell, a)[R'_{ag}(\vec{q}, s, \ell, a, \vec{q}', s', \ell') + \gamma v^\rho(\vec{q}', s', \ell')]$. By using the equality between R'_{ag} and R''_{ag} we have: $v^\rho(\vec{q}, s, \ell) = \sum_{\vec{q}', s', \ell'} P(\vec{q}', s', \ell'|\vec{q}, s, \ell, a)[R''_{ag}(\vec{q}, s, a, \vec{q}', s') + \gamma v^\rho(\vec{q}', s', \ell')]$. On the other hand, observe that we can compute $\vec{q}'$ from $\vec{q}$ and ℓ', that is we do not need ℓ. Hence: $P(\vec{q}', s', \ell'|\vec{q}, s, \ell, a) = P(\vec{q}', s', \ell'|\vec{q}, s, a)$. So we can write: $v^\rho(\vec{q}, s, \ell) = \sum_{\vec{q}', s', \ell'} P(\vec{q}', s', \ell'|\vec{q}, s, a)[R''_{ag}(\vec{q}, s, a, \vec{q}', s') + \gamma v^\rho(\vec{q}', s', \ell')]$. At this point, we see that v^ρ does not depend from ℓ, hence we can safely drop ℓ as argument for v_ρ. Indeed, we get: $v^\rho(\vec{q}, s) = \sum_{\vec{q}', s'}[R''_{ag}(\vec{q}, s, a, \vec{q}', s') + \gamma v^\rho(\vec{q}', s')] \sum_{\ell'} P(\vec{q}', s', \ell'|\vec{q}, s, a)$ and by marginalizing the distribution $P(\vec{q}', s', \ell'|\vec{q}, s, a)$ over ℓ', we get: $v^\rho(\vec{q}, s) = \sum_{\vec{q}', s'} P(\vec{q}', s'|\vec{q}, s, a)[R''_{ag}(\vec{q}, s, a, \vec{q}', s') + \gamma v^\rho(\vec{q}', s')]$. This is Bellman's equation for M_{ag}^q, hence the thesis. □

This theorem provides us with a technique to learn the optimal policy for RL with LTL$_f$/LDL$_f$ restraining specification by making minimal intervention to the learning agent: essentially we need to feed it with the rewards r_i at suitable times, and we need to allow the learning agent to keep track of the stage of satisfaction of the restraining bolt formulas by feeding it with new features for $Q_1, \ldots, Q_n$.

Implementation and Examples

Implementation of agents learning policies with restraining specifications is performed by assuming a learning phase in simulation and an execution phase on the real world. The learning phase is obtained by combining three software components: 1) a simulator of the dynamic system, 2) a restraining bolt (RB) process, 3) a reinforcement learning (RL) agent. All these components are modular (i.e., they can be properly connected each other or replaced by other similar components). The simulator is responsible for computing

Figure 2: Experimental scenarios: BREAKOUT, SAPIENTINO, COCKTAILPARTY

the evolution of the dynamic system under study: it receives decisions (actions to be executed) by the RL agent and communicates: i) the current state of the system to both the RL agent and the RB process, and ii) the current reward value to the RL agent. The RB process receives the current state from the simulator, evaluates the LTL$_f$/LDL$_f$ formulas denoting the restraining specifications and sends to the RL agent an encoding of the progress of the DFA representing the formulas and reward values associated to their evolution. Finally, the RL agent receives the simulator state, the RB state, and the rewards and decides the actions to be executed, while computing an optimal policy. By using such a simulator, the RL agent can learn a policy that maximizes the cumulative discounted reward taking into account both rewards from the environment and rewards from the RB. In general, when enough training is allowed, the computed policy, when executed on the real world, will satisfy the RB specifications.

As mentioned, the RL agent and the RB process have different sensors to perceive different aspects of the state of the world (or of the simulator). So we assume that they are implemented with real sensors (when attached to the real world) and corresponding virtual sensors (when attached to the simulator). We also assume that the simulator is able to model all the relevant evolutions of the world that are needed to learn the specific task with restraining specifications.

Next we show the implementation of such components in three examples (Figure 2). The first one uses a video-game simulator, while the other two consider robotic tasks and their corresponding models in a simulator. The core software for the RL agent and for the RB process are domain-independent, while the (virtual) sensors and the LTL$_f$/LDL$_f$ specifications are domain-dependent. 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).

To speed up learning, the implementation of the bolt monitors the progress of the DFA corresponding to the restraining specifications and applies a kind of reward shaping by exploiting the DFA structure[9]. Through reward shaping we can anticipate part of the reward coming from temporal specifications without waiting for the formulas to become true.

Each experiment (i.e., a sequence of episodes to learn a policy) terminates after a time limit that is different for each problem (see next sections) and chosen to guarantee that a policy consistent with the specifications is always found, although in general not optimal. All the problems described

[9]Reward shaping is not described here for lack of space

below have been solved with n-step Sarsa algorithm, configured with $\gamma = 0.999$, $\epsilon = 0.2$, $n = 100$. The trend of the solutions is anyway not sensitive to these parameters[10].

Algorithms have been implemented as single-thread non-optimized Python procedures, in a modular and abstract way to operate on every problem. More details about the experimental configurations, source code of the implementation allowing for reproducing the results contained in this paper, and videos of the found policies are available in `www.diag.uniroma1.it/restraining-bolt`.

Breakout. BREAKOUT has been widely used to demonstrate RL approaches. The goal of the agent is to control the paddle in order to drive a ball to hit all the bricks in the screen. In this example, we have considered two agents with different abilities: MOVE: the agent moves sideways to bounce the ball; MOVE +FIRE: the agent can both move and fire straight up to remove bricks. Agent's state representation uses the following features: f_x: x position of the paddle; $f_{bx}, f_{by}, f_{dx}, f_{dy}$: position and direction of movement of the ball[11]. Reward is given to the agent when a brick is hit. With this specification a RL algorithm can find a policy to remove all the bricks and complete the game for both the agents.

Restraining bolt. We want to provide the agents with the following specification: *the bricks must be removed from left to right*, i.e., all the bricks in column i must be removed before completing any other column $j > i$. This specification can be expressed with an LTL$_f$/LDL$_f$ formula and to evaluate such a formula, the bolt needs a representation $f_{r(i,j)}$ of the status of each brick $r_{i,j}$ (present or removed). The agents, after receiving in input from the bolt an encoding of the status of the LTL$_f$/LDL$_f$ formula and associated rewards, can use the same RL algorithm to learn a new policy that will complete the task (i.e., remove all the bricks) following the restraining bolt specification (i.e., from left to right).

Notice that the same restraining bolt is applied to the two different agents and they will both learn the behavior specified by the LTL$_f$/LDL$_f$ formula, obviously with different policies. Rows 1 and 2 in Figure 3 show the results of two experiments in the Breakout scenario with the following configurations: Breakout 4x6 MOVE + FIRE (5 minutes), Breakout 4x5 MOVE (1 hour). Left plots show the average reward over the number of iterations, while right plots show the score (i.e., number of columns correctly broken) of the best policy computed so far (i.e., the results obtained in runs without exploration). The figures show how the agent is able to progressively learn how to progress over the states of the DFA corresponding to the LTL$_f$/LDL$_f$ specification. Similar results are obtained in different configurations (e.g., different sizes of the bricks). reported in the columns is encoded with the first letter being either M for MOVE and F for FIRE actions available, and the second letter being either L for LOCAL and G for GLOBAL for the sensor modality.

Sapientino. SAPIENTINO Doc is an educational game for 5-8 y.o. children where a small mobile robot has to be programmed to visit specific cells in a 5x7 grid. Some cells con-

[10]See more results on the web site.

[11]Other state representations are also suitable to learn the task.

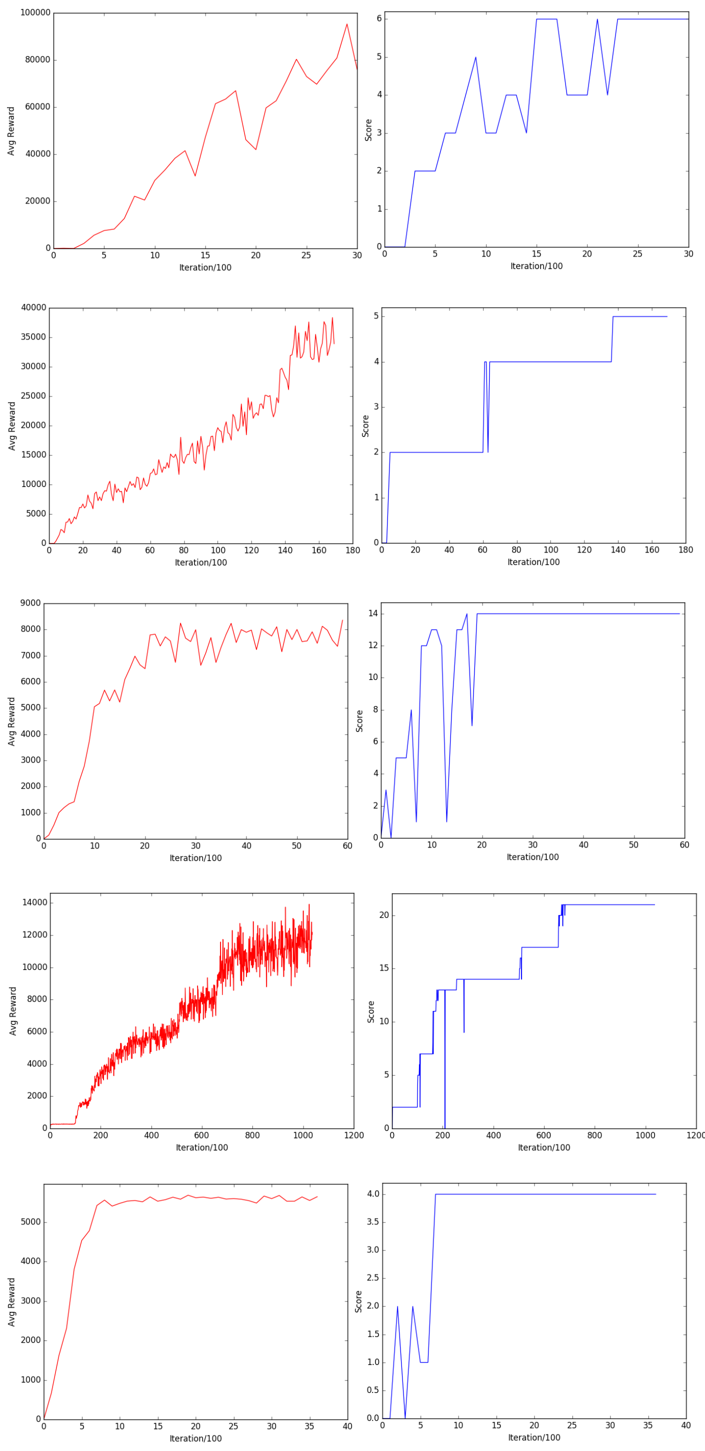

Figure 3: Average reward and scores over number of iterations. Row 1: Breakout MOVE + FIRE 4x6 bricks (5 minutes); Row 2: Breakout MOVE only 4x5 bricks (1 hour); Row 3: Sapientino S2 OMNI (3 minutes); Row 4: Sapientino S3 DIFFERENTIAL (1 hour); Row 5: Cocktail Party (3 minutes).

tain concepts that must be matched by the children (e.g., a colored animal, a color, the first letter of the animal's name), while other cells are empty. The robot executes sequences of actions given in input by children with a keyboard on the robot's top side. During execution, the robot moves on the grid and executes an action (actually a *bip*) to announce that the current cell has been reached (this is called a *visit* of a cell). A pair of consecutive visits are correct when they refer to cells containing matching concepts. As in the real game, we consider a 5x7 grid with 7 triplets of colored cells, each triplet representing three matching concepts. *State representation* is defined by the following features: f_x, f_y, f_θ reporting the pose of the agent in the grid. In this scenario, we consider two different agents: OMNI: omni-directional movements (actions: up, down, left, right), DIFFERENTIAL: differential drive (actions: forward, backward, turn left, turn right). With this specification, the agent can just learn how to move in the grid, but it cannot match related concepts.

Restraining bolt. Consider now the specifications S2: *visit at least two cells of the same color for each color, in a given order among the colors* (the order of the colors is predefined: first C_1, then C_2, and so on) and S3: *visit all the triplets of each color, in a given order among the colors*. The following additional features are needed to express and evaluate the corresponding formula: f_b reporting that a *bip* action has just been executed and f_c reporting the color of the current cell.

The restraining specifications for these games can be expressed with LTL_f formulas. A fragment of LTL_f formula for the first game relative to the first color C_1 is

$$\begin{array}{l}\neg bip\,\mathcal{U}(\bigvee_{j=1,2,3} cell_{C_1,j} \wedge bip) \wedge \\ \bigwedge_{j=1,2,3} \Box(cell_{C_1,j} \wedge bip \rightarrow \circ\Box(bip \rightarrow \neg cell_{C_1,j})) \wedge \\ \bigvee_{j=1,2,3} \Box(cell_{C_1,j} \wedge bip \rightarrow \circ(\neg bip\,\mathcal{U} \bigvee_{k\neq j} cell_{C_1,k} \wedge bip)\end{array}$$

For other colors C_{i+1}, we use a similar formula, but requiring that $\bigvee_{j=1,2,3} cell_{C_i,j} \wedge bip$ has already been satisfied.

Two agents and two restraining bolts can be combined to form 4 different learning situations. We show only two of them. Rows 3 and 4 of Figure 3 show the agents' learning ability (score = 14 for S2 specs, score = 21 for the S3 specs). Similar results are obtained in different configurations.

Cocktail party. For a service robot involved in a cocktail party we consider a representation of the state in terms of robot's pose and objects' (drinks and snacks) and people's location. The agent can move in the environment, grasp and deliver items to people, and get a reward when a delivery task is completed. The robot has no sophisticated people perception capabilities, and no memory is available in the underlying MDP modeling the domain, so the robot cannot get information about individual people or remember who received what. The robot in this scenario will just learn how to bring one item to any person (choosing the shortest path).

Restraining bolt. Consider the following specification: *serve exactly one drink and one snack to every person, but do not serve alcoholic drinks to minors*. As in the previous examples, the restraining bolt works on separate features, namely identity, age and received items[12] and uses an $\text{LTL}_f/\text{LDL}_f$ formula to model this specification. operating scenario. We assume the map of the environment to be known, people sitting at tables in predefined known positions and locations of snack and drink items also known. From these information we can instantiate a simulator for the robot to navigate in this environment and reach the different locations[13].

For learning this task, we considered a problem with two people and two different kinds of drinks and snacks (4 tasks to be executed in total) and we implemented an abstract simulator reproducing the scenario of RoboCup@Home competition. The results of learning the restrained task in the simulator are depicted in Row 5 of Figure 3 (score = 4 means that the 2 persons have received one drink and one snack each). As shown, after about 1 minute of simulation[14], the RL agent converged to a policy satisfying the RB specifications.

Minecraft. As an example of our approach's modularity, we used the same agent of SAPIENTINO in a MINECRAFT scenario. Here the agent has to accomplish 10 tasks (described with non-Markovian rewards via an $\text{LTL}_f/\text{LDL}_f$ formula). The two agents share the same state representation S but differ in the action set A, the fluent configurations $\mathcal{L}$, and the component progressing the DFAs. Results (not shown here) confirm that a *general-purpose* agent can learn several tasks by only receiving information from its restraining bolt.

Conclusions

We have shown how to perform RL with $\text{LTL}_f/\text{LDL}_f$ restraining specifications by resorting to typical RL techniques based on MDPs. Notably, we have shown that the features needed to evaluate $\text{LTL}_f/\text{LDL}_f$ formulas can be kept separated from those directly accessible to the learning agent.

Our work can be ascribed to that part of research generated by the urgency of providing safety guarantees to AI techniques based on learning (Amodei et al. 2016; Hadfield-Menell et al. 2017; Orseau and Armstrong 2016). In particular, it shares similarities with recent work on constraining the RL agent to satisfy certain safety conditions (Wen, Ehlers, and Topcu 2015; Achiam et al. 2017; Alshiekh et al. 2018). There are however important differences. First, in enforcing the restraining bolt we consider the learning agent essentially as a black box. That is, the restraining bolt does not need to know the internals S of the learning agent, and specifies the desired constraints using only its world features $\mathcal{L}$. On the other hand, we do not guarantee the satisfaction of the restraining bolt constraints during training, as in (Achiam et al. 2017). In fact, differently from (Wen, Ehlers, and Topcu 2015; Alshiekh et al. 2018), we do not guarantee the hard satisfaction of constraints even after training. After all "You can't teach pigs to fly"! and we may very well ask to do so in our restraining bolts, being these completely unrestricted in the selection of world features and in the kind of formulas they specify over such features. If we want to check formally

[12]In practice, services like Microsoft Cognitive Services Face API can be integrated into the bolt to provide this information.

[13]Specifically, we used Stage simulator in ROS with standard navigation stack.

[14]This time can be drastically reduced using optimized code.

that the optimal policy satisfies the restraining bolt specification, we first need to model how actions affect the restraining bolt's features $\mathcal{L}$, i.e., we need to link the learning agent's features S to $\mathcal{L}$, and then we can use, e.g., model checking. Notably, for doing RL we do not need to specify such a link, but we can simply allow the (possibly simulated) world to act as the link, in line with what advocated, e.g., in (Brooks 1991), and very differently from what typically considered in knowledge representation (Reiter 2001).

Apart from restraining bolts, the interest in having separate representations is manifold. The learning agent feature space can be designed separately from the features needed to express the goal, thus promoting *separation of concerns* which, in turn, facilitates the design, providing for *modularity* and *reuse* of representations (the same agent can learn from different bolts and the same bolt can be applied to different agents). Also, a reduced agent's feature space allows for realizing *simpler agents* (think, e.g., of a mobile robot platform, where one can avoid specific sensors and perception routines), while preserving the possibility of acting according to complex declarative specifications which cannot be represented in the agent's feature space. We plan to investigate this separation further in the future.

Acknowledgements Work partially funded by Sapienza University of Rome, under projects "Immersive Cognitive Environments" and "Data-awaRe Automatic Process Execution", and by the European Union's Horizon 2020 research and innovation program under grant agreement N. 825619.

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained policy optimization. In *ICML*, 22–31.

Alberto; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017. Non-markovian rewards expressed in LTL: Guiding search via reward shaping. In *SOCS*, 159–160.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe reinforcement learning via shielding. In *AAAI*, 2669–2678.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *CoRR* abs/1606.06565.

Bacchus, F.; Boutilier, C.; and Grove, A. 1996. Rewarding behaviors. In *AAAI*, 1160–1167.

Baier, J. A.; Fritz, C.; Bienvenu, M.; and McIlraith, S. A. 2008. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI*.

Brafman, R. I.; De Giacomo, G.; and Patrizi, F. 2018. LTL$_f$/LDL$_f$ non-markovian rewards. *AAAI*.

Brooks, R. A. 1991. Intelligence without representation. *Artif. Intell.* 47(1-3):139–159.

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017. Decision-making with non-markovian rewards: From LTL to automata-based reward shaping. In *RLDM*, 279–283.

De Giacomo, G., and Rubin, S. 2018. Automata-theoretic foundations of FOND planning for LTLf and LDL$_f$ goals. In *IJCAI*.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*.

Gretton, C. 2007. Gradient-based relational reinforcement learning of temporally extended policies. In *ICAPS*, 168–175.

Gretton, C. 2014. A more expressive behavioral logic for decision-theoretic planning. In *PRICAI*, 13–25.

Hadfield-Menell, D.; Dragan, A. D.; Abbeel, P.; and Russell, S. J. 2017. The off-switch game. In *IJCAI*, 220–227.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2018. Teaching multiple tasks to an RL agent using LTL. In *AAMAS*, 452–461.

Lacerda, B.; Parker, D.; and Hawes, N. 2014. Optimal and dynamic planning for markov decision processes with co-safe LTL specifications. In *IROS*, 1511–1516.

Lacerda, B.; Parker, D.; and Hawes, N. 2015. Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications. In *IJCAI*.

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming* 31.

Littman, M. L.; Topcu, U.; Fu, J.; Jr., C. L. I.; Wen, M.; and MacGlashan, J. 2017. Environment-independent task specifications via GLTL. *CoRR* abs/1704.04341.

Littman, M. L. 2015. Programming agent via rewards. In *Invited talk at IJCAI*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Orseau, L., and Armstrong, S. 2016. Safely interruptible agents. In *UAI*.

Pnueli, A. 1977. The temporal logic of programs. In *FOCS*.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Rabin, M. O., and Scott, D. 1959. Finite automata and their decision problems. *IBM J. Res. Dev.* 3:114–125.

Reiter, R. 2001. *Knowledge in Action*. MIT Press.

Silver, D.; amd Karen Simonyan, J. S.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of go without human knowledge. *Nature* 550:354–359.

Simsek, Ö., and Barto, A. G. 2006. An intrinsic reward mechanism for efficient exploration. In *ICML*, volume 148 of *ACM International Conference Proceeding Series*, 833–840. ACM.

Slaney, J. K. 2005. Semipositive LTL with an uninterpreted past operator. *Logic Journal of the IGPL* 13(2):211–229.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning - an introduction*. MIT Press.

Thiébaux, S.; Gretton, C.; Slaney, J. K.; Price, D.; and Kabanza, F. 2006. Decision-theoretic planning with non-markovian rewards. *J. Artif. Intell. Res. (JAIR)* 25:17–74.

Wen, M.; Ehlers, R.; and Topcu, U. 2015. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *IROS*, 4983–4990.

Whitehead, S. D., and Lin, L.-J. 1995. Reinforcement learning of non-markov decision processes. *Artificial Intelligence* 73(1):271 – 306. Computational Research on Interaction and Agency, Part 2.