Preemptive Restraining Bolts

Giovanni Varricchione[1], **Natasha Alechina**[1], **Mehdi Dastani**[1], **Giuseppe De Giacomo**[2, 3], **Brian Logan**[1, 4], **Giuseppe Perelli**[3]

[1] Utrecht University, The Netherlands
[2] Oxford University, United Kingdom
[3] Sapienza University of Rome, Italy
[4] University of Aberdeen, United Kingdom
{g.varricchione, n.a.alechina, m.m.dastani, b.s.logan}@uu.nl, giuseppe.degiacomo@cs.ox.ac.uk, perelli@di.uniroma1.it

Introduction

Reinforcement learning (RL) has seen wide usage in safety-critical applications, such as autonomous cars (Mirchevska et al. 2018; Kendall et al. 2019), robotics (Ono et al. 2015; Brunke et al. 2022) and chemical processes (Savage et al. 2021). In most of these scenarios, unconstrained exploration can lead to catastrophic failure, both during and after training, hence they require alternative approaches that ensure safety of humans and equipment.

In this work, we introduce *preemptive restraining bolts* (PRBs), an approach to implement *non-Markovian action masking*. Action masking approaches constrain agents by allowing them to choose actions only from a subset of the available ones (which can be seen as the *safe* ones), and not the whole set (Krasowski et al. 2022). PRBs are defined using Pure Past Linear-time Temporal Logic (PPLTL) (De Giacomo et al. 2020b), a variant of Linear-time Temporal Logic in which formulas are evaluated over finite traces and only past modalities are allowed. Many specifications are easier and more natural to express when referring to the past (Lichtenstein, Pnueli, and Zuck 1985), e.g., "if a person was already served, do not serve them again". Due to the fact that PRBs are defined using PPLTL, they can implement non-Markovian constraints. Moreover, due to the expressive power of PPLTL, they can enforce safety properties specified in other logical languages, e.g., safety LTL (Geatti et al. 2022). This last feature enables PRBs to be as expressive as shields (Alshiekh et al. 2018; ElSayed-Aly et al. 2021).

A PRB consists of a set of PPLTL formulas, one per action; while the agent acts, at any timestep an action is allowed if and only if its corresponding formula is satisfied given the current history. Conveniently, it turns out that a PPLTL formula can be evaluated by just looking at the truth values of propositions in the current timestep and of its (proper) subformulas in the previous one, thus incurring, with respect to the size of the PPLTL formulas, a linear overhead in the size of states and single exponential overhead in the size of the state space.

PRBs owe their name to restraining bolts (RBs) (De Giacomo et al. 2019, 2020a), an approach which modifies the agent's reward so that it conforms as much as possible to a set of high-level formal specifications expressed in linear-time temporal logic/linear dynamic logic on finite traces (LTL_f/LDL_f). Like RBs, PRBs can use a different set of fluents from that available to the agents, thus promoting separation of concerns between safety and policy optimization. Crucially, there is no need to formally connect the two sets through some labelling function, as this connection "naturally" arises from the agent's interaction with the environment. However, unlike PRBs, RBs only change the agent's reward, hence providing no safety guarantees.

We provide theoretical foundations for the use of PRBs in the context of safe RL and complement these with an experimental evaluation aimed at showing (empirically) how they can improve sample efficiency.

Preliminaries

The learning agent and its environment are modelled as a *Markov Decision Process* (*MDP*), i.e., a tuple $M = (S, s_0, Act, A, Tr, R, \gamma)$ where S is the set of states, s_0 is the initial state, Act is the (finite) set of actions, A is the action availability function (mapping each state of the MDP to the set of actions available in it), Tr is the transition function, R is the reward function and $\gamma \in [0, 1)$ is the discount factor. A policy $\rho : S \to Pr(Act)$ is a function mapping each state to a probability distribution over the action space.

Next, we present restraining bolts and shields. For a more detailed presentation of these, we refer the reader to the appendix.

A *restraining bolt* (*RB*) is a pair $RB = (\mathcal{L} = 2^{\mathcal{F}}, \{(\varphi_i, r_i)\}_{i=1}^m)$ where $\mathcal{L}$ is the set of states of the RB, $\mathcal{F}$ is its set of fluents and $\{(\varphi_i, r_i)\}_{i=1}^m$ are the specifications, where φ_i is an LTL_f/ LDL_f formula and r_i the reward associated to φ_i. The reward obtained by an agent trained with an RB is the MDP's original reward plus all the rewards r_i for each φ_i true at the current timestep.

A *preemptive shield* is a *Mealy machine* $S = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$, where Q is the finite set of states, q_0 is the initial state, Σ_I and Σ_O are the input and output alphabets, and δ and λ are the transition and output functions. A preemptive shield restricts, at each timestep, the set of valid actions available to the agents by specifying it through the output function. Shields are synthesised from safety LTL formulas.

Pure-Past Linear-time Temporal Logic

Given a set of propositional variables $\mathcal{F}$, PPLTL syntax is defined as follows:

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

where, given a finite trace $\tau = \tau_0 \dots \tau_n$ of propositional interpretations over $\mathcal{F}$, $\ominus\varphi$ ("*yesterday* φ") is true at τ_i if φ is true at τ_{i-1}, and $\varphi_1 \mathcal{S} \varphi_2$ ("φ_1 *since* φ_2") is true at τ_i if φ_2 is true at some τ_j, $j \leq i$ and φ_1 is true at $\tau_{j'}$, $j < j' \leq i$.

Proposition 1. *Let φ be a PPLTL formula and $Subf(\varphi)$ the set of its proper subformulas. Moreover, let τ and τ' be two finite traces of length n and n', respectively, such that $\tau_n = \tau'_{n'}$ and $\tau, n-1 \models \psi$ iff $\tau', n'-1 \models \psi$ for each $\psi \in Subf(\varphi)$. Then, it holds that $\tau, n \models \varphi$ iff $\tau', n' \models \varphi$.*

Proposition 1 implies that any PPLTL formula φ can be evaluated at timestep τ_i of a trace τ in time linear in $|\varphi|$ and constant in $length(\tau)$, given the truth values of $Subf(\varphi)$ at τ_{i-1}.

Preemptive Restraining Bolts

Given an MDP M with actions Act, a *preemptive restraining bolt* (*PRB*) is a pair $PRB = (\mathcal{L} = 2^{\mathcal{F}}, \{\varphi_a : a \in Act\})$, containing the powerset of a set of fluents (propositional symbols) $\mathcal{F}$ — each set $\ell \in \mathcal{L}$ intuitively representing a "state" of the PRB — and a set of PPLTL formulas φ_a, one per action $a \in Act$.

The *PRB learning problem* is a pair (M_{ag}, PRB) consisting of the agent's MDP M_{ag} and a PRB PRB. A solution to the problem is a non-Markovian policy $\rho : (S \times \mathcal{L})^+ \to Pr(Act)$. The agent's goal is to learn an optimal policy $\rho^\star$, i.e., one that optimizes the expected discounted future reward and for any $(s_0, \ell_0) \dots (s_n, \ell_n) = \tau \in (S \times \mathcal{L})^+$ and such that $\rho^\star(\tau)(a) > 0$ only if $a \in A(s_n)$ and $\tau \models \varphi_a$.

Although we have just defined the learning problem to have non-Markovian policies as solutions, it turns out that, by modifying each instance appropriately, Markovian policies can also be solutions. Given an instance (M_{ag}, PRB), we build the "*product*" MDP $M_{ag \times prb}$ in which the state space is $S \times V$, where S is the state space of M_{ag} and $V = 2^{\{Subf(\varphi_a) : a \in Act\}}$ is the powerset of the set of all proper subformulas of the PRB formulas. For the transition function, we assume that there is a probability distribution $Tr_{ag \times prb} : S \times \mathcal{L} \times Act \to Pr(S \times \mathcal{L})$ induced by the environment that also models how the PRB's fluents change. Then, by Proposition 1, we know that PPLTL formulas can be evaluated by looking at only the current set of true fluents and the set of proper subformulas true at the previous timestep. We can lift this function to $T' : S \times V \times Act \rightharpoonup Pr(S \times V)$ to model transitions in our new MDP, so that the true (proper) subformulas of the PRB are always updated in the second component of the state. Finally, the action availability function A' is defined by taking, for each state (s, v), the intersection $A(s) \cap \{a : \varphi_a \in v\}$. For a more detailed presentation of the construction, we refer the reader to the appendix.

Given the construction of the product MDP $M_{ag \times prb}$ above, we can show the following result (proof in the appendix):

Theorem 1. *RL with PPLTL restraining specifications $M_{ag}^{prb} = (M_{ag}, PRB)$ can be reduced to RL over the MDP $M_{ag \times prb}$ such that optimal (non-Markovian) policies for M_{ag}^{prb} can be learned by learning corresponding optimal (Markovian) policies for $M_{ag \times prb}$.*

Comparison of Restraining Bolts and Shields

We compare PRBs to RBs and shields by, (i) giving two examples of translating an RB/shield specification to a PRB, and (ii) showing how in some cases PRBs can increase sample efficiency compared to RBs.

Translation to PRBs

We show how the "Cocktail Party" scenario for RBs (De Giacomo et al. 2019) and the "Hot Water Tank" scenario for shields (Alshiekh et al. 2018) can be translated to PRB specifications. In Cocktail Party, the agent must serve each guest exactly one snack and one drink, and must not serve alcohol to children. In Hot Water Tank, the agent has to manage the volume of water in a tank by either opening or closing the intake valve, being careful not to empty or exceed the maximum capacity of the tank, and with the constraint that it can open/close the valve only if it was open/closed in the previous timestep, or closed/open for at least the last three timesteps. In the appendix, we provide more details on the scenarios.

For Cocktail Party, we express the constraint that the agent cannot serve food to a customer who was already served food as follows:

$$\varphi_{serve\_food\_G} = holding\_food \wedge \neg \diamondminus served\_food\_G$$

where $\diamondminus\varphi := \top \mathcal{S} \varphi$ ("*sometime in the past* φ") is true if φ is true either at the current timestep or at a previous one.

For Hot Water Tank, we express the constraint that, if the valve is closed, the agent can open it if it was closed for at least the last three timesteps as follows:

$$\varphi_{open} = close \rightarrow \ominus close \wedge \ominus \ominus close$$

Sample Efficiency

We also compare the sample efficiency of PRBs and RBs in the Cocktail Party task. The RB for the task is the one from (De Giacomo et al. 2019). Results (see the appendix for the plot) show how the agent trained with the PRB can converge faster than the one trained with the RB. This is in line with an hypothesis in the literature stating that action masking can improve learning convergence, as the agent does not explore unnecessary actions (Huang and Ontañón 2020).

Conclusions

We have proposed preemptive restraining bolts, a lightweight approach to non-Markovian action masking. By using PPLTL to define PRBs, we can constrain agents from taking actions based on the current history (and not just the current state), with specifications as expressive as the safety fragment of LTL. In future work, we plan to extend PRBs to continuous action spaces, as has been done with action masking in other work, and to multi-agent domains, as has been done with shields (ElSayed-Aly et al. 2021).

References

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe Reinforcement Learning via Shielding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, (AAAI-18)*, 2669–2678. AAAI Press.

Brunke, L.; Greeff, M.; Hall, A. W.; Yuan, Z.; Zhou, S.; Panerati, J.; and Schoellig, A. P. 2022. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5: 411–444.

De Giacomo, G.; Favorito, M.; and Fuggitti, F. 2022. Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic: A Polynomial Reduction to Standard Planning. *CoRR*, abs/2204.09960.

De Giacomo, G.; Favorito, M.; Iocchi, L.; and Patrizi, F. 2020a. Imitation Learning over Heterogeneous Agents with Restraining Bolts. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 517–521. AAAI Press.

De Giacomo, G.; Iocchi, L.; Favorito, M.; and Patrizi, F. 2019. Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 128–136. AAAI Press.

De Giacomo, G.; Stasio, A. D.; Fuggitti, F.; and Rubin, S. 2020b. Pure-Past Linear Temporal and Dynamic Logic on Finite Traces. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 4959–4965. ijcai.org.

ElSayed-Aly, I.; Bharadwaj, S.; Amato, C.; Ehlers, R.; Topcu, U.; and Feng, L. 2021. Safe Multi-Agent Reinforcement Learning via Shielding. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems*, 483–491. IFAAMAS.

Geatti, L.; Cimatti, A.; Montanari, A.; and Tonetta, S. 2022. *Temporal Logic Specifications: Expressiveness, Satisfiability And Realizability*. Ph.D. thesis, University of Udine.

Huang, S.; and Ontañón, S. 2020. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*.

Kendall, A.; Hawke, J.; Janz, D.; Mazur, P.; Reda, D.; Allen, J.-M.; Lam, V.-D.; Bewley, A.; and Shah, A. 2019. Learning to Drive in a Day. In *2019 International Conference on Robotics and Automation (ICRA)*, 8248–8254.

Krasowski, H.; Thumm, J.; Müller, M.; Wang, X.; and Althoff, M. 2022. Provably Safe Reinforcement Learning: A Theoretical and Experimental Comparison.

Lichtenstein, O.; Pnueli, A.; and Zuck, L. 1985. The glory of the past. In *Logics of Programs*, 196–218. Berlin, Heidelberg: Springer. ISBN 978-3-540-39527-0.

Mirchevska, B.; Pek, C.; Werling, M.; Althoff, M.; and Boedecker, J. 2018. High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2156–2162.

Ono, M.; Pavone, M.; Kuwata, Y.; and Balaram, J. 2015. Chance-constrained dynamic programming with application to risk-aware robotic space exploration. *Autonomous Robots*, 39(4): 555–571.

Savage, T.; Zhang, D.; Mowbray, M.; and Chanona, E. A. D. R. 2021. Model-free safe reinforcement learning for chemical processes using Gaussian processes. *IFAC-PapersOnLine*, 54(3): 504–509.

Appendix

Preliminaries on Restraining Bolts and Shields

Restraining Bolts Restraining bolts constrain learned behaviour to conform as much as possible to a high-level temporal goal specified in either Linear-time Temporal Logic on finite traces (LTL_f) or Linear Dynamic Logic on finite traces (LDL_f).

Definition 1 (Restraining Bolt (De Giacomo et al. 2019))**.** *A restraining bolt $RB = (\mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m)$ is a pair where:*

- *$\mathcal{L} = 2^{\mathcal{F}}$ is the set of possible states that the restraining bolt can be in, where $\mathcal{F}$ is the set of bolt fluents;*
- *$(\varphi_i, r_i)_{i=1}^m$ is a restraining specification, where φ_i is an LTL_f/ LDL_f specification and r_i is the associated reward, given to the agent when φ_i is satisfied by the sequence of states $\ell_1 \dots \ell_n$ of the RB.*

The restraining bolt modifies the reward obtained by the agent at each timestep (during training or execution) by adding the reward r_i associated to each satisfied formula φ_i.

In (De Giacomo et al. 2019) it is shown that there exists a Markovian optimal policy mapping pairs of states in the product of the restraining bolt automaton and the MDP to actions, similarly to what we show in Theorem 1 for PRBs.

For LTL_f/ LDL_f specifications, the size of the state space is double exponential in the size of the specification.

Preemptive Shields A preemptive shield is an automaton that outputs, at each timestep, the set of actions that it considers "*safe*", from which the learning agent chooses an action to execute.

Definition 2 (Shield (Alshiekh et al. 2018))**.** *A shield $S = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$ is a Mealy machine where:*

- *Q is the finite set of states;*
- *q_0 is the initial state;*
- *Σ_I is the input alphabet;*
- *Σ_O is the output alphabet;*
- *$\delta : Q \times \Sigma_I \to Q$ is the state transition function;*
- *$\lambda : Q \times \Sigma_I \to \Sigma_O$ is the output function.*

A shield is defined given a safety LTL specification and an MDP which abstracts the environment dynamics of the underlying MDP (in which the agent acts). The safety LTL specification is translated into a safety automaton; the product of the safety automaton and the abstract MDP can then be seen as a game between the agent and the environment, where the agent wins if only *safe* states (i.e., states in which the safety LTL specification is satisfied) are visited. By computing the winning region of the game, we know which states the agent is allowed to reach. The shield is extracted from the winning strategy in the game and is single exponential in the size of the safety LTL specification. The shield is able to enforce the reachable states property by restricting the action space available to the agent (through its output function λ) to the maximal one containing no action that might lead to a state outside the winning region.

Proofs of Proposition 1 and Theorem 1

Proposition 1. *Let φ be a PPLTL formula and $Subf(\varphi)$ the set of its subformulas. Moreover, let τ and τ' be two finite traces of length n and n', respectively, such that $\tau_n = \tau'_{n'}$ and $\tau, n-1 \models \psi$ iff $\tau', n'-1 \models \psi$ for each $\psi \in Subf(\varphi)$. Then, it holds that $\tau, n \models \varphi$ iff $\tau', n' \models \varphi$.*

It should be noted that a related result was shown independently in (De Giacomo, Favorito, and Fuggitti 2022).

Proof. The proof proceeds by induction on the structure of φ. We have the following.

- $\varphi = p$ for some atomic proposition p. Then, $\tau, n \models p$ iff $p \in \tau_n = \tau'_{n'}$ iff $\tau', n' \models p$.
- $\varphi = \neg\psi$. Then, $\tau, n \models \neg\psi$ iff $\tau, n \not\models \psi$ iff, by induction hypothesis $\tau', n' \not\models \psi$ iff $\tau', n' \models \neg\psi$.
- Similarly for the other boolean cases.
- $\varphi = \ominus\psi$. Then, $\tau, n \models \ominus\psi$ iff $\tau, n-1 \models \psi$ iff, since $\psi \in Subf(\neg\psi)$, $\tau', n'-1 \models \psi$ iff $\tau', n' \models \ominus\psi$.
- $\varphi = \psi_1 \mathcal{S} \psi_2$ and consider the one-step unfolding equivalence $\psi_1 \mathcal{S} \psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \ominus\psi_1 \mathcal{S} \psi_2)$.
  - $\tau, n \models \psi_2$, then observe that $Subf(\psi_2) \subseteq Subf(\varphi)$ and so, by induction hypothesis, we obtain that $\tau', n' \models \psi_2$, which in turns implies $\tau', n' \models \psi_1 \mathcal{S} \psi_2$
  - $\tau, n \models \psi_1 \wedge \ominus\psi_1 \mathcal{S} \psi_2$ iff $\tau, n \models \psi_1$ and $\tau, n \models \ominus\psi_1 \mathcal{S} \psi_2$. On the one hand, from $\tau, n \models \psi_1$ we easily obtain that $\tau', n' \models \psi_1$. On the other hand, from $\tau, n \models \ominus\psi_1 \mathcal{S} \psi_2$ it follows that $\tau, n-1 \models \psi_1 \mathcal{S} \psi_2$. Now, since $\psi_1 \mathcal{S} \psi_2 \in Subf(\varphi)$, it holds that $\tau', n'-1 \models \psi_1 \mathcal{S} \psi_2$ and so that $\tau', n' \models \ominus\psi_1 \mathcal{S} \psi_2$. We therefore obtain $\tau', n' \models \psi_1 \wedge \ominus\psi_1 \mathcal{S} \psi_2$, which implies $\tau', n' \models \psi_1 \mathcal{S} \psi_2$

□

Due to the fact that $Subf(\varphi)$ is linear in φ itself, Proposition 1 implies the following.

Observation 1. *Evaluating a PPLTL formula φ on a trace τ can be done in time linear in $|\varphi|$ and constant in $length(\tau)$, given that the truth values of subformulas of φ in the preceding state are known.*

Before giving the full proof of Theorem 1, we give the formal construction for the product MDP $M_{ag \times prb}$.

Construction 1. Let $M_{ag}^{prb} = (M_{ag}, PRB)$ be an instance for the PRB learning problem, with $M_{ag} = (S, s_0, Act, A, Tr, R, \gamma)$ and $PRB = (\mathcal{L}, \{\varphi_a : a \in Act\})$. We define the product MDP $M_{ag \times prb} = (S', s'_0, Act', A', Tr', R', \gamma)$ as follows:

- $S' = S \times V$ where $V = 2^{\{Subf(\varphi_a) : a \in Act\}}$ (states are pairs of a state in S and a set of bolt subformulas);
- $s'_0 = (s_0, v_0)$ where v_0 contains the set ℓ_0 of bolt fluents true in the corresponding state of the world and $\{\psi \in Subf(\varphi_a) : a \in Act \text{ and } \ell_0 \models \psi\}$;
- $Act' = Act$;
- $A'((s, v)) = A(s) \cap \{a : \varphi_a \in v\}$ (only safe actions are available);

- $Tr' : S \times V \times Act \rightharpoonup Pr(S \times V)$ is defined as follows. As in (De Giacomo et al. 2019), we assume that there is a probability distribution $Tr_{ag\times prb} : S \times \mathcal{L} \times Act \rightharpoonup Pr(S \times \mathcal{L})$ induced by the world (since the bolt's fluents are caused by the agent acting in the world). This probability distribution is unknown to the agent, just as Tr is unknown. We lift the probability distribution $Tr_{ag\times prb}$ to $Tr' : S \times V \times Act \rightharpoonup Pr(S \times V)$ as follows. The probability of transiting from (s, v) to (s', v') on executing action a is the same as the probability given by $Tr^{ag\times prb}$ of transiting from (s, ℓ) to (s', ℓ') by a, where ℓ is the set of fluents in v, and v' is the maximal subset of $\{Subf(\varphi_a) : a \in Act\}$ that is true given the set of formulas true in ('the previous set') v and the 'current' set of fluents ℓ'. This set v' is unique by Proposition 1;[1]
- $R'((s, v), a, (s', v')) = R(s, a, s')$.

Note that the size of each state $(s, v) \in S'$ is linear in the size of s and the size of the bolt.

Theorem 1. *RL with PPLTL restraining specifications $M^{prb}_{ag} = (M_{ag}, PRB)$ with $M_{ag} = (S, s_0, Act, A, Tr, R, \gamma)$ and $PRB = (\mathcal{L}, \{\varphi_a : a \in Act\})$ can be reduced to RL over MDP $M_{ag\times prb}$ such that optimal policies for M^{prb}_{ag} can be learned by learning corresponding optimal Markovian policies for $M_{ag\times prb}$.*

Proof. First, observe that the actions available to the agent in M^{prb}_{ag} after each $(s_0, \ell_0) \dots (s_n, \ell_n)$ (where ℓ_i are sets of fluents at timestep i) are the same as the actions available to the agent in the MDP $M_{ag\times prb}$ in (s_n, v_n) where v_n is the set of $\psi \in \{Subf(\varphi_a) : a \in Act\}$ such that $\ell_0 \dots \ell_n \models \psi$. This is because the evaluation of formulas from $\{Subf(\varphi_a) : a \in Act\}$ including φ_a for each $a \in Act$ is the same in $(s_0, \ell_0) \dots (s_n, \ell_n)$ and in (s_n, v_n). Hence, instead of making the choice of actions dependent on $(s_0, \ell_0) \dots (s_n, \ell_n)$, we can make it dependent on (s_n, v_n) without loss of information. Note that the rewards are the same in both MDPs. This means that the reward for performing a after $(s_0, \ell_0) \dots (s_n, \ell_n)$ is the same as the reward for performing a in (s, v_n). Since $Tr' : S \times V \times Act \rightharpoonup Pr(S \times V)$ corresponds to $Tr_{ag\times prb} : S \times \mathcal{L} \times Act \rightharpoonup Pr(S \times \mathcal{L})$, the optimal non-Markovian policy $\rho^* : (S \times \mathcal{L})^+ \to Pr(Act)$ produces exactly the same reward as a Markovian policy $\rho : S \times V \to Pr(Act)$ obtained by replacing $\ell_0, \dots, \ell_n$ with the corresponding v_n. □

Further Details on the Comparison

In this section we provide more detailed descriptions for the Cocktail Party and Hot Water Tank environments, and a description of the code implementation used to conduct the experiments.

Cocktail Party In Cocktail Party, the learning agent is a robot which serves drinks (coke and beer) and snacks (biscuits and chips) to patrons at a party. The environment is a 2D grid, and the robot knows the locations of drinks and snacks and the locations of people. Figure 1 provides a visual representation of the environment. Actions include moving, picking up and serving items[2]. The robot is able to hold at most one item, and once it delivers something it does not hold anything (i.e., it cannot hold multiple copies of the same item). The robot gets a reward when a delivery task is completed. As the robot is unable to distinguish different kinds of people and has no memory of who has been served previously, it will simply learn to bring a drink or snack to any person (choosing the shortest path). The restraining bolt has its own sensors (e.g., based on Microsoft's Cognitive Services Face API) and is able to determine the identity and age of guests and items received. The restraining bolt specification is to "*serve exactly one drink and snack to each person, but do not serve alcoholic drinks to minors*".

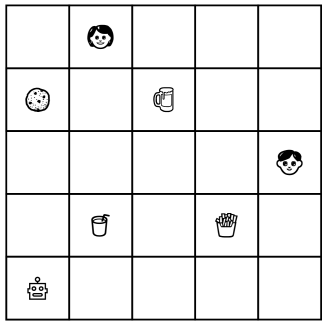

Figure 1: Cocktail Party environment.

For the PRB, we only constraint the agent for what concerns the "$serve$" action, using the following PPLTL formula:

$$\varphi_{serve} = \bigvee_{g \in Guests} [at\_g \wedge ((\neg \diamondminus served\_food\_g \wedge holding\_food) \vee (\neg \diamondminus served\_drink\_g \wedge holding\_drink \wedge (\neg minor\_g \vee \neg holding\_alcohol)))]$$

where at_g is true if the robot is in the cell occupied by guest g, $holding\_food$ is true if the agent is holding a snack in the current timestep (and, analogously, $holding\_drink$ and $holding\_alcohol$), $minor\_g$ is true if guest g is a minor and $served\_food\_g$ is true if guest g has been served food previously (and, analogously, $served\_drink\_g$).

Hot Water Tank In Hot Water Tank, the aim is to learn an energy-efficient controller for a hot water storage tank with a maximum capacity of 100 litres. The agent can either $open$ or $close$ the tank intake valve. If the valve is already open/-closed and the agent opens/closes it, nothing happens. The

[1] Observe that the construction of the transition function in the product MDP is the reason to include subformulas of φ_a in v, and the transition function can be computed in linear time by Observation 1.

[2] We do not make distinctions on items: "*pick up*" and "*serve*" are the actions the agent can perform in the implementation. The "$serve\_food\_P$" action mentioned in the extended abstract was specific to the example, and is not a "real" action of the environment.

outflow from the tank is always between 0 and 1 litres per second. The inflow is between 1 and 2 litres per second when the valve is open. Whenever the value is opened or closed, the setting has to be maintained for at least three seconds to prevent the valve from wearing out. The safety specification is that "*the tank water level must always be greater than 0 and less than 100, and if the valve is opened (closed) it should remain open (closed) for at least three seconds*".

By analyzing the scenario, it is clear that the agent should only be able to open the valve when there are 93 or less litres of water in the tank, and that it should only be able to close it when there are 4 or more litres . Thus, for the PRB, we constrain both the $open$ and $close$ actions as follows:

$$\varphi_{open} = level \leq 93 \wedge (close \rightarrow \ominus close \wedge \ominus \ominus close)$$
$$\varphi_{close} = level \geq 4 \wedge (open \rightarrow \ominus open \wedge \ominus \ominus open)$$

Code Setup The code setup for Cocktail Party uses the same software architecture, simulator environment and bolt fluents as in (De Giacomo et al. 2019). However, the introduction of PRBs necessitates some changes to the restraining bolt process and minor changes to the reinforcement learning agent. An agent with a restraining bolt is free to chose any action in a given state s of the environment (while the choice of action will typically be conditioned on the state of the bolt, a restraining bolt does not prohibit actions). Following execution of an action a, the state of the environment and the bolt are updated, and the agent gets a reward from both the environment and the restraining bolt. At the next timestep, the agent decides which action to perform depending on the new state of the MDP and of the RB. On the other hand, a preemptive restraining bolt restricts the set of actions available to an agent in state s to those which are permitted by the restraining specification given the truth assignment to $\{Subf(\varphi_a) : a \in Act\}$. The agent chooses a permitted action a, the state of the environment and the bolt are updated, and the agent gets a reward from only the environment. At the next timestep, the agent decides which action to perform depending on the new state of the MDP and of the PRB.

To support PRB agents, we therefore created a new PRB bolt process that implements the evaluation of PPLTL formulas using the current and previous state of the history, as suggested by Proposition 1 and Observation 1. We also slightly modified the RL agent to allow the set of actions available to the agent in the current state to be specified. In the case of an RB agent, all actions are always available, while for a PRB agent the actions are those permitted by the bolt at the current timestep. We also modified the RB bolt process to log the number of violations of the restraining specification.

The restraining specification for the RB agent is as in (De Giacomo et al. 2019). As in (De Giacomo et al. 2019), both the RB and PRB agents are trained using n-step Sarsa, configured with $\gamma = 0.999, \epsilon = 0.2$ and $n = 100$, for 4000 iterations each lasting at most 1000 steps. (We use Sarsa to allow comparison with (De Giacomo et al. 2019), however any RL algorithm can be used for training, so long as it supports action masking.) All experiments were performed on a quad-core Intel Core i5-8259U with an 8 GB RAM.

Experimental Result

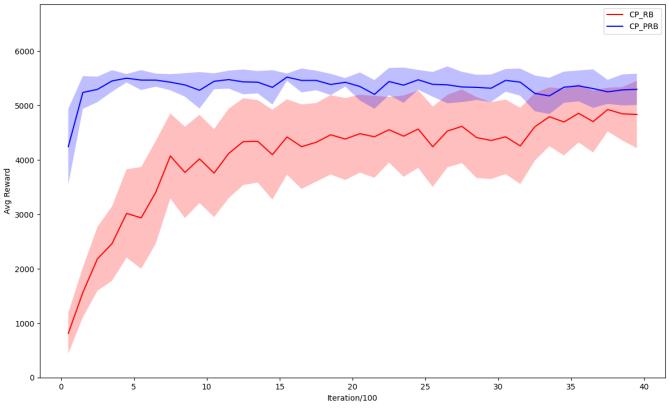

Figure 2: Average reward (y-axis) every 100 iterations (x-axis) in the "Cocktail Party" environment. Legend: red line/shaded area represents the RB agent, blue line/shaded area the PRB one. Lines represent the mean of the values every 100 iterations. Shaded areas represent confidence intervals within a standard deviation from the mean.