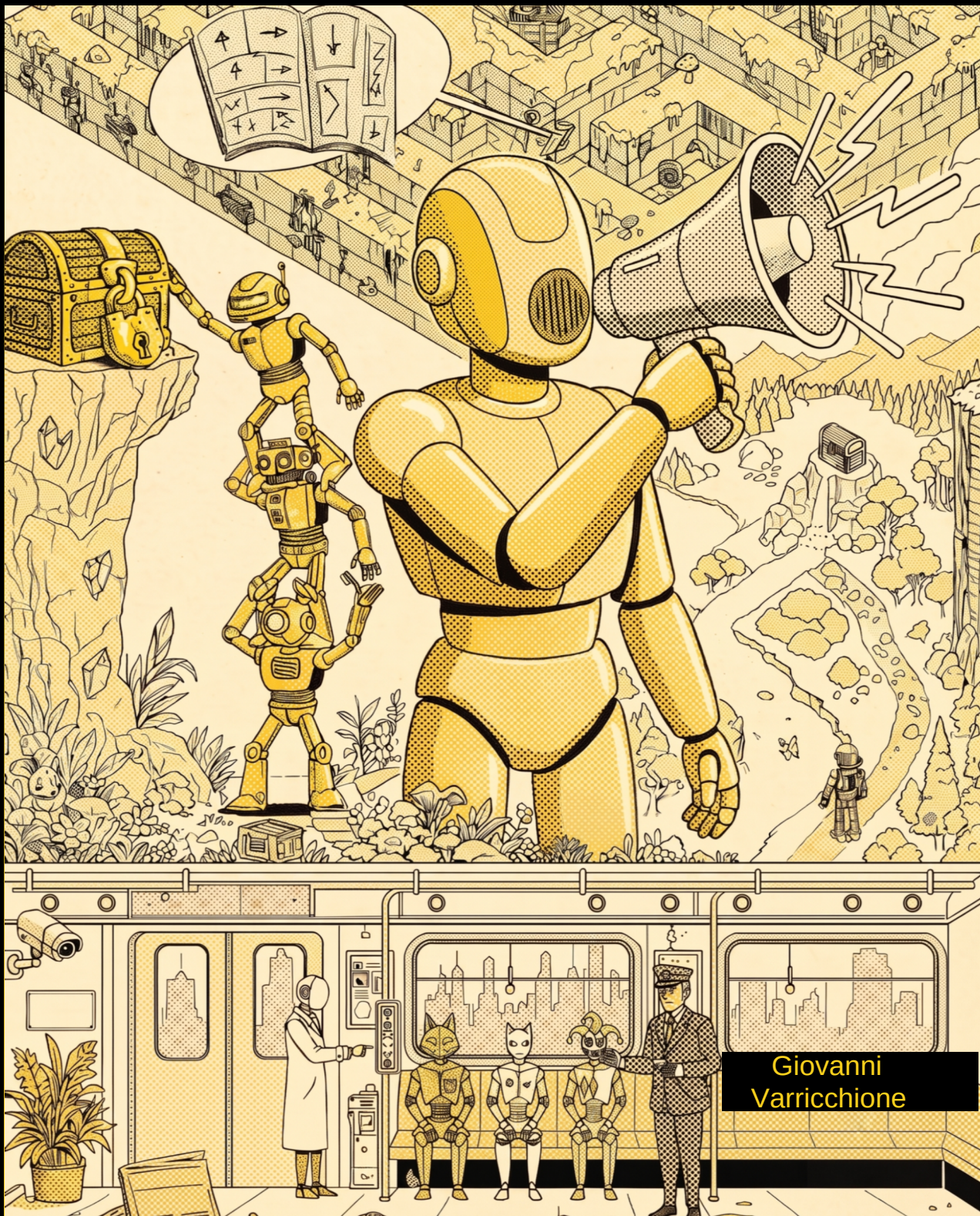


Utrecht University Repository

Title	Declarative Specifications for Efficient and Safe Reinforcement Learning
Authors	Varricchione, Giovanni
Published in	SIKS Dissertation Series
Publication Date	2026-08-28
Link	https://dspace.library.uu.nl/handle/1874/489138
Citation	Varricchione, G 2026, 'Declarative Specifications for Efficient and Safe Reinforcement Learning', Doctor of Philosophy, Universiteit Utrecht, Utrecht. https://doi.org/10.33540/3683
Versions / License	Publisher version
Rights	https://www.uu.nl/en/university-library/license-and-reuse-conditions

Declarative Specifications for Efficient and Safe Reinforcement Learning



Declarative Specifications for Efficient and Safe Reinforcement Learning

Giovanni Varricchione

DOI: 10.33540/3683
ISBN: 978-94-6537-863-3
Printing: Ridderprint



**Utrecht
University**



SIKS Dissertation Series No. 2026-32

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Copyright © 2026 Giovanni Varricchione

Cover design by Cristian Cianfarani and Giovanni Varricchione

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form without the written permission of the copyright owner.

Declarative Specifications for Efficient and Safe Reinforcement Learning

Declaratieve Specificaties voor Efficiënte en Veilige Reinforcement Learning

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit Utrecht
op gezag van de
rector magnificus, prof. dr. ir. W. Hazeleger,
ingevolge het besluit van het College voor Promoties
in het openbaar te verdedigen op
vrijdag 28 augustus 2026 des middags te 4.15 uur

door

Giovanni Varricchione

geboren op 17 september 1996
te Benevento, Italië

Promotoren: Prof. dr. M. M. Dastani
Prof. dr. B. S. Logan

Beoordelingscommissie: Prof. dr. S. Abeln
Prof. dr. A. Montanari
Prof. dr. A. Plaat
Prof. dr. M. T. J. Spaan
Dr. S. Wang



Contents

1	Introduction	1
2	Preliminaries	7
2.1	Reinforcement Learning	7
2.2	Reinforcement Learning Algorithms	10
2.3	Reward Machines	20
2.4	Linear-time Temporal Logic	28
3	Related Works	31
3.1	Reward Machines	31
3.2	Declarative Approaches for Sample Efficient RL	35
3.3	Declarative Approaches for Safe RL	37
4	Maximally Permissive Reward Machines	39
4.1	Introduction	40
4.2	Maximally Permissive Reward Machines	45
4.3	Empirical Evaluation	47
4.4	Conclusions	52
4.5	Supplementary Material	53
5	Synthesising Reward Machines for Cooperative Multi-Agent RL	57
5.1	Introduction	57
5.2	Preliminaries	60
5.3	Synthesising Markov Games With Reward Machines	70
5.4	Evaluation	77
5.5	Conclusions	89
6	Pushdown Reward Machines for Reinforcement Learning	91
6.1	Introduction	92
6.2	Pushdown Reward Machines	93

6.3	When Are k -Policies Optimal?	96
6.4	Comparison With CRAs	97
6.5	Exploiting Counterfactual Experiences	99
6.6	Experimental Evaluation	100
6.7	Conclusions	109
6.8	Supplementary Material	110
7	Pure-Past Action Masking	123
7.1	Introduction	123
7.2	Preliminaries	125
7.3	Pure-Past Linear-time Temporal Logic	127
7.4	Pure-Past Action Masking	128
7.5	Comparison with Shields	132
7.6	Experimental Evaluation	134
7.7	Conclusions	137
7.8	Supplementary Material	138
8	Conclusions and Future Directions	143
8.1	Contributions of the Dissertation	143
8.2	Future Research	147
8.3	Conclusion	149
	Bibliography	169
	Samenvatting	171
	SIKS Dissertation Series	173
	Curriculum Vitae	191
	Acknowledgements	193



1 Introduction

In the development of artificial intelligence (AI), decision making is a crucial aspect to enable the deployment of AI agents in the real world. In order for an agent to achieve its objectives, it must be able to make decisions on how to act. When we do not have full knowledge of the environment in which the agent will act, defining how the agent should take decisions can be a difficult task, as we do not know precisely what will be the effects of each action. Reinforcement learning (RL) (Sutton and A. G. Barto, 2018), one of the main machine learning paradigms, is specifically designed to train agents to make decisions on how to act in order to achieve given goals, under the assumption that the environment is not fully known a-priori. Agents trained with reinforcement learning are trained to maximize the expected cumulative sum of a scalar reward signal, which acts as a proxy of the objective. At each step, the agent first picks an action based on its current perception of the environment, commonly referred to as “*environment state*”, and then, after performing the action, it observes the new state and the reward signal.

There are two underlying assumptions in reinforcement learning. First, the agent has limited or no knowledge of the environment. This means that the agent does not fully know how the environment state is updated at each step (i.e., it does not know the “*environment’s dynamics*”), nor how its actions modify the environment, nor what kind of reward signal it will observe after performing an action from a state. Thus, the agent is supposed to learn by a process of trial-and-error, discovering which are the best actions to accrue higher rewards in each state. Second, the environment is assumed to be *Markovian*. Hence, the next state of the environment depends only on the current environment state and the action that the agent performs. The reward signal is also assumed to be Markovian: the agent’s reward depends only on the current state of the environment, the action that the agent takes, and the next state of the environment.

The two assumptions on which reinforcement learning relies make it a versatile and useful framework to train agents to achieve objectives. The fact that the agent does not have full knowledge of the environment is crucial when, for example, we do not know how to mathematically model the environment in which the agent acts, or when we simply do not know in which environment the agent will act. Thanks to the Markovian assumption, the probabilities of transitions between states and

of the reward signals depend just on the current state instead of the entire history so far. In this way, we can consider a “simplified” version of the environment for which it is possible to easily provide a mathematical model. Due to the versatility of reinforcement learning and to the advent of deep learning, and hence also of deep reinforcement learning (Plaat, 2022), there have been several advancements and successes in AI. In deep RL, neural networks are used to encode the agent’s behaviour; as deep RL approaches employ neural networks, they are also referred to as “*neural*” RL approaches. Neural RL approaches include several algorithms, such as PPO (Schulman, Wolski et al., 2017) and SAC (Haarnoja et al., 2018), which allowed agents to achieve remarkable results in complex Markovian domains without having any a-priori knowledge. Reinforcement learning and its neural approaches were central in the development of agents that can beat professional human players in a variety of games (D. Silver et al., 2016; Berner et al., 2019), design computer chips (Mirhoseini et al., 2021), and design new drugs (Zhavoronkov et al., 2019). Last, but not least, reinforcement learning has also played a crucial role in the development of large language models (Ouyang et al., 2022; K. Liu et al., 2025).

Nevertheless, the two assumptions that are usually made in RL can hinder the training of agents, especially when they should learn to comply with safety constraints, or limit the set of tasks that agents can learn to achieve. For example, as noted in (Singh, 1992a; Singh, 1992b; X. Li, Vasile et al., 2017; Toro Icarte, Klassen, Valenzano and McIlraith, 2018; Camacho, Toro Icarte et al., 2019; De Giacomo, Iocchi et al., 2019; Shah et al., 2020; M. Hasanbeig, Kroening et al., 2022; Toro Icarte, Klassen, Valenzano and McIlraith, 2022; Toro Icarte, 2022), the reward function can be revealed to the agent. As argued in (Toro Icarte, 2022), when the reward function is coded by a (human) designer who has full knowledge of it, it can be reasonable to allow the agent to also access it. Similarly, in some cases, we might have partial knowledge of the environment’s dynamics or of how the agent is supposed to achieve the task. For example, when training agents to control robots, we could provide the agent with the laws of physics that affect how the robot moves (T. Silver et al., 2019; Ramesh and Ravindran, 2023). Akin to the argument made by Toro Icarte for reward functions, if one is to code part of the environment’s dynamics, then such dynamics could also be revealed to the agent. There are also scenarios in which we want our RL-trained agent to adhere to a set of safety constraints. In RL, it is generally not possible to ensure that the agent will respect these constraints, especially during training (Krasowski, Thumm et al., 2023). As the agent learns by a process of trial-and-error and it is not provided with any information about the environment, the agent must, at some point, violate the safety constraints in order to learn how to not violate them. However, if we could provide the agent with some a-priori information on the constraints, similarly to how part of the rewards and dynamics can be revealed, the agent could leverage such information to learn safe behaviours more easily. Finally, there are also domains in which the environment’s dynamics and/or the task might not be Markovian (Gaon and Brafman, 2020). For example, the task might require the agent to perform specific sequences of actions over time, like preparing and delivering a coffee to a customer only after receiving an order from such customer. Although in principle it is possible to augment the set of information available to the agent to make the dynamics and rewards Markovian again (Bacchus et al., 1996), recently developed

methods are more sophisticated and use structures like automata to achieve better performance (De Giacomo, Iocchi et al., 2020; Toro Icarte, Klassen, Valenzano and McIlraith, 2022). Developing approaches that specifically do not assume that the environment is Markovian could be crucial for training agents that can achieve more complex tasks while reducing training times.

In this dissertation, we will specifically consider cases where the designer can provide the learning agent with (some) a-priori knowledge of the environment's rewards and dynamics and where the Markovian assumption fails. In order to do so, we will employ *declarative* AI. In declarative AI approaches, agents are defined by explicitly modelling how they should behave and what goals they should achieve. This is done by defining behaviours and goals using, e.g., rules expressed via logic formulas, or formal mathematical objects such as automata, or via planning. By defining agents using human-readable formalisms, declarative AI supports interpretability and one can easily understand why an agent has acted in a certain manner. Declarative AI has its own shortcomings: it cannot handle scenarios that do not comply with the agent's specifications, and the agent is not able to correct by itself a wrong specification which can be due to, for example, wrong assumptions of the designer about the environment (McCarthy and P. J. Hayes, 1981). However, it can easily lend itself to the purposes we need it for in this dissertation. Indeed, unlike neural approaches which struggle to exploit a-priori knowledge (Marcus, 2018)¹, declarative AI provides several tools which can be leveraged to encode a-priori knowledge. As we will see, the designer's knowledge of the environment can be easily encoded by using mathematical formalisms typical of declarative AI, such as logic or automata. By doing so, it is then possible to exploit results and ideas from declarative AI to improve training in RL, especially for non-Markovian tasks. Declarative AI can also be used to easily encode, thanks to logic-based rules, safety constraint which the agent must always satisfy. Further, by using human-readable formalisms, declarative AI allows to define approaches for RL that are more human-interpretable and transparent. This is also a clear benefit when compared against neural approaches, as the (deep) neural networks used in them are *black boxes* and thus difficult to interpret (Marcus, 2018).

The thesis of this dissertation is thus the following:

Thesis. *Declarative AI techniques can be leveraged in reinforcement learning to: (i) improve efficiency and quality of resulting policies for non-Markovian tasks, and (ii), guarantee the satisfaction of safety constraints both at training and execution time.*

We split the contributions of this dissertation in two, based on points (i) and (ii) from the thesis above.

For point (i), we will employ *reward machines* (Toro Icarte, Klassen, Valenzano and McIlraith, 2018; Toro Icarte, Klassen, Valenzano and McIlraith, 2022; Toro Icarte, 2022). Reward machines can be used to train agents to achieve tasks that require to perform specific sequences of actions over time, thus enabling the expression of tasks where the Markovian assumption does not hold. Since reward machines are

¹In Section 3.6 of (Marcus, 2018), Marcus explicitly discusses how prior knowledge is difficult to integrate with pure neural approaches as “the dominant approach in deep learning is hermeneutic, in the sense of being self-contained and isolated from other, potentially usefully knowledge”.

finite state automata, they can be defined either by hand or via formal specifications. We will give new ways of defining reward machines from declarative specifications, using planning for single-agent RL and logic for multi-agent RL. We will also modify reward machines and equip them with a *pushdown stack*, allowing to define a strictly larger set of non-Markovian reward functions compared to that definable by “vanilla” reward machines. As we will see, thanks to our reward machines-based approaches, we will be able to train agents in achieving complex non-Markovian tasks while also simplifying task definition and, in some cases, shortening training time.

For point (ii), we will shift our focus to *safe* RL, where agents are trained to achieve tasks while adhering to safety constraints. We will specifically demand agents to always be *provably safe*, following the definition provided in (Krasowski, Thumm et al., 2023). Thus, we will provide formal guarantees that the agents will never violate the safety constraints, not even during training. To achieve this, we will employ Pure-Past Linear-time Temporal Logic (PPLTL), a temporal logic which formulas describe properties over sequences of (past) observations. We will constrain the agent by using PPLTL formulas to limit the set of allowed actions at a specific timestep. Furthermore, by using PPLTL, we can define constraints that are non-Markovian, as they can depend on the entire history and not just the current state.

Reinforcement learning has had an important role in the recent developments of AI, thanks to its versatility and how it can be easily used to train agents. Although research in RL has led to outstanding results in AI, there are still issues that can be addressed by incorporating ideas from declarative AI; the works in this dissertation follow this observation. We hope that, together with its related literature, this dissertation will provide further insights into how to leverage declarative AI for RL, and that it will inspire further research in this area.

Publications

The contents of the chapters in this dissertation are based on the following publications (listed in chronological order of publication):

1. Giovanni Varricchione, Natasha Alechina, Mehdi Dastani and Brian Logan. ‘Synthesising Reward Machines for Cooperative Multi-Agent Reinforcement Learning’. In: *Proceedings of the 20th European Conference on Multi-Agent Systems (EUMAS)*. 2023, pp. 328–344.
2. Giovanni Varricchione, Natasha Alechina, Mehdi Dastani, Giuseppe De Giacomo, Brian Logan and Giuseppe Perelli. ‘Pure-Past Action Masking’. In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Technical Track on Safe, Robust and Responsible AI. 2024, pp. 21646–21655.
3. Giovanni Varricchione, Natasha Alechina, Mehdi Dastani and Brian Logan. ‘Maximally Permissive Reward Machines’. In: *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI)*. 2024, pp. 1181–1188.
4. Giovanni Varricchione, Toryn Q. Klassen, Natasha Alechina, Mehdi Dastani, Brian Logan and Shiela A. McIlraith. ‘Pushdown Reward Machines for Re-

inforcement Learning’. In: *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 2025, pp. 566–576.

5. Giovanni Varricchione, Natasha Alechina, Mehdi Dastani and Brian Logan. ‘Synthesising Reward Machines for Cooperative Multi-Agent Reinforcement Learning’. In: *Journal of Artificial Intelligence Research* 85.38 (2026).

Dissertation Structure

We conclude this introductory chapter by presenting the structure of the dissertation, shortly summarizing each of the following chapters.

- Chapter 2 introduces the background notions that are necessary for the contents in this dissertation. We give a formal introduction to reinforcement learning and present two learning algorithms that have been used for the experimental results of the works included in this dissertation. Then, we present reward machines and how they can be exploited to improve training in RL. We conclude by giving an introduction to Linear-time Temporal Logic.
- Chapter 3 comprises of a collection of works that are related to those in this dissertation. We first include works in the reward machines literature, both in the single-agent and multi-agent settings. Then, we consider declarative AI-based approaches that have been used to improve training of RL agents, and finally declarative AI-based approaches that were specifically concerned with training *safe* RL agents.
- Chapter 4 presents (Varricchione, Alechina, Dastani and Logan, 2024), the first work we include in this dissertation. We show how *maximally permissive reward machines* (MPRMs) can be obtained by considering all possible plans that can achieve a task. As MPRMs are defined from multiple plans, they allow agents to achieve tasks in a more versatile and efficient way. We conclude by showing empirically how MPRMs affect convergence speed and policy quality when compared to reward machines obtained from individual plans.
- Chapter 5 presents (Varricchione, Alechina, Dastani and Logan, 2023) and (Varricchione, Alechina, Dastani and Logan, 2026). In it, we move from the single-agent case to the multi-agent one. We show how RMs for multi-agent RL can be obtained from specifications given in a fragment of Alternating-time Temporal Logic, and how agents can be trained in practice to achieve tasks in a fully cooperative setting compared against reward machines defined by hand.
- Chapter 6 presents (Varricchione, Klassen et al., 2025). In this work we have augmented reward machines and equipped them with a pushdown stack, defining *pushdown reward machines* (pdRMs). As pdRMs have access to a pushdown stack, they can model a strictly larger set of non-Markovian reward functions compared to vanilla RMs. We define two kinds of policies that a pdRM-trained agent can learn, and provide a counterfactual algorithm specifically designed

for pdRMs. We then provide an experimental evaluation to show how agents can be trained in achieving various tasks using pdRMs.

- Chapter 7 presents (Varricchione, Alechina, Dastani, De Giacomo et al., 2024), the last work we include in this dissertation. By leveraging Pure-Past Linear-time Temporal Logic (PPLTL), we define *pure-past action masks* (PPAMs). PPAMs can be used to constrain action selection by associating to each action that the agent can perform a PPLTL formula. At any timestep, the agent is allowed to perform an action if and only if its corresponding PPLTL formula is true. We relate PPAMs to shields (Alshiekh et al., 2018), another approach in the RL literature which guarantees that agents adhere to a given set of safety constraints at any time. We conclude by showing how PPAMs can be used in practice, and how they ensure that agents do not violate safety.
- Chapter 8 concludes by summarising the works presented in the previous chapters, and by presenting possible future directions of research which are in line with the spirit of this dissertation.

2 Preliminaries

We will now present the necessary background to make this work self-contained. We will formally define the RL problem, and present the RL algorithms that were used in the works which this thesis comprises of. As a large portion of this thesis regards reward machines, an automata-based approach for RL, we will also introduce and formally define these. We conclude the chapter by introducing linear-time temporal logic, a logic which is central to several chapters in the dissertation.

2.1 Reinforcement Learning

In reinforcement learning (Sutton and A. G. Barto, 1998; Sutton and A. G. Barto, 2018), one of the three fundamental machine learning paradigms, agents learn behaviours to achieve tasks in (possibly unknown) environments. As a proxy to learn such behaviours, agents actually learn to maximise a reward signal provided by the environment. The main underlying assumption in RL is that the agent does not have full knowledge of the so-called “*dynamics*” of the environment, i.e., how the environment is updated at each timestep and what reward signal it will output. Hence, agents need to *explore* the environments and learn from *experiences* in order to attain such behaviours.

Environments in RL are modelled as *Markov decision processes* (MDPs) (Puterman, 2014). In what follows, for a given set X we denote with $\Delta(X)$ the set of all probability distributions over X . A probability distribution for a set X is any function $\delta : X \rightarrow [0, 1]$ satisfying the condition that $\sum_{x \in X} \delta(x) = 1$.

Definition 2.1 (Markov Decision Process). A Markov decision process M is a tuple $\langle S, \mu, S^F, A, T, R, \gamma \rangle$ where S is the non-empty set of states, $\mu \in \Delta(S)$ is the initial state distribution, $S^F \subseteq S$ is the (possibly empty) set of *terminal* states, A is the non-empty set of actions, $T : S \times A \rightarrow \Delta(S)$ is the state transition function, $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor.

Before continuing, we note that agents can either have full knowledge of the current MDP state or not. In the latter, the agent is not able to fully discern different states of the MDP, implying that it has access to a *partial* representation of the current

state. This is formalized with the concept of *partially observable MDPs* (POMDPs) (Spaan, 2012), which is the model of environments in *partially observable* RL. However, for what concerns this dissertation, we will always assume that the agent has full knowledge of the current MDP state.

To improve readability, we will write $T(s' \mid s, a)$ to denote the probability of transitioning from state s to state s' when action a is performed in s , i.e., the value $T(s, a)(s')$. With an abuse of notation we might also define MDPs without the initial state distribution and/or the set of terminal states S^F (note that, if the set of final states is not given, then we assume that it is the empty set).

At each timestep, the MDP M is in some state $s \in S$. As the agent takes some action $a \in A$, the MDP updates its state to s' by sampling it from $T(s, a)$, i.e., $s' \sim T(s, a)$, and outputs the reward signal $r = R(s, a, s')$. The observed tuple (s, a, s', r) is then referred to as an “*experience*” or “*experience sample*”. If the MDP reaches a terminal state $s^F \in S^F$, then the agent stops acting as the MDP halts.

During training, the agent acts in the MDP, which is always initialised in some state s_0 sampled according to the initial state distribution μ . Central to training is the concept of “*episode*”, i.e., a sequence of experience samples starting from an initial state of the MDP and ending either in a terminal state (if there is any), or after a number of timesteps (usually referred to as the “*horizon*”) has elapsed. If no horizon is specified, then the agent can potentially act in the MDP indefinitely, in a so-called “*continuous task*”, and episodes can be infinite sequences of experience samples. In the rest of this thesis, unless specified, we will assume that the agents perform “*episodic tasks*”, i.e., tasks where a finite horizon is specified.

By using the experiences samples gathered during the training episodes, the agent can learn a *policy*.

Definition 2.2 (Policy). For an MDP $M = \langle S, \mu, S^F, A, T, R, \gamma \rangle$, a policy is a function $\pi : S \rightarrow \Delta(A)$ that returns, for each state of M , a probability distribution over the set of actions of M .

Similarly to transitions in MDP, we will write $\pi(a \mid s)$ to denote the probability of taking action a in state s , i.e., the value $\pi(s)(a)$. To improve readability and clarity, we will also write $\pi(\cdot \mid s)$ to denote the action probability distribution of π in state s , i.e., $\pi(s)$.

Agents can have both *stochastic* and *deterministic* policies: the latter are those for which, for any state s of the MDP, there is one action a such that $\pi(a \mid s) = 1$.

Policies represent agents’ behaviours and induce a probability distribution over state-action-state-reward sequences in the MDP. Intuitively, one can think of any such sequence as an attempt by the agent to achieve the task. With each sequence the agent is also endowed with an associated *discounted return* G_t for a given timestep t of the sequence. Given a sequence of reward $R_1, \dots, R_t, \dots$, the discounted return G_t is defined as follows:

$$G_t := r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad (2.1)$$

As MDPs and policies can be non-deterministic, discounted returns can change from run to run. Therefore to compare policies we use the *expected discounted reward*,

defined starting from some state s at any given timestep t :

$$\mathbb{E}_\pi [G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{j=0}^{\infty} \gamma^j r_{t+j} \mid S_t = s \right] \quad (2.2)$$

In $\mathbb{E}_\pi [G_t \mid S_t = s]$, we treat G_t as a random variable which values are all the possible returns that an agent can obtain while following policy π from the starting state s ; the probability of G having some value g is the sum of the probabilities of all the traces which return is exactly g (as defined in Eq. (2.1)). When the policy is clear from the context, we will drop it from $\mathbb{E}_\pi$.

The expected discounted future reward that we have just defined is also used as the *state-value function* of a policy π , defined as $v_\pi(s) := \mathbb{E}_\pi [G_t \mid S_t = s]$. When π is clear from the context, we will write just v instead of v_π . Crucially, the state-value function (of any policy π in any MDP) satisfies a recursive relationship connecting the value of any MDP state s to the value of all its successors:

$$v(s) = \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} T(s' \mid s, a) [R(s, a, s') + \gamma v(s')] \quad (2.3)$$

Equation (2.3) is also called “*Bellman equation for v_π* ” (Bellman, 1958; Bellman, 1966). The intuition behind the Bellman optimality equation is that the value of any state is obtained by averaging, across all possible states that can be reached at the next timestep, the immediate reward obtained by transitioning to the next state and the value of such next state.

Similarly to states, we can define the value of a state-action pair s, a as the expected discounted future reward obtained if the agent takes action a in state s and from the successive timestep keeps following the given policy π . We thus define the so-called “*action-value function*”:

$$q_\pi(s, a) := \mathbb{E} [G_t \mid S_t = s, A_t = a] = \mathbb{E} \left[\sum_{j=0}^{\infty} \gamma^j r_{t+j} \mid S_t = s, A_t = a \right] \quad (2.4)$$

When clear from the context, we will also drop the policy π from q_π . Similarly to the state-value function, the action-value function can also be defined in a recursive manner:

$$q(s, a) = \sum_{s' \in S} T(s' \mid s, a) \left[R(s, a, s') + \gamma \sum_{a' \in A} \pi(a' \mid s') q(s', a') \right] \quad (2.5)$$

thus obtaining the “*Bellman equation for q_π* ”.

As we have stated in the beginning of this section, the goal in reinforcement learning is for the agent to learn a policy that allows it to achieve the task. In MDPs, the reward function acts as a proxy of the task. Thus, the agent’s objective is to learn a policy that maximizes the reward it obtains. However, since the reward the agent obtains by following a policy is not necessarily deterministic (recall how policies and the dynamics of MDPs can both be non-deterministic), policies should actually

maximize the *expected discounted reward*. This last observation allows us to order policies, by considering the values that they achieve in the states of the MDP: a policy π is “better” than some other policy π' , denoted with $\pi \geq \pi'$, whenever for each state of the MDP s we have that $v_\pi(s) \geq v_{\pi'}(s)$.

We thus define a policy to be “*optimal*” whenever it is maximal with respect to the ordering $\geq$ we have just defined. It should be noted that there always exists such an optimal policy. Optimal policies thus achieve the so-called “*optimal state-value function*” v^* :

$$v^*(s) := \max_{\pi} v_{\pi}(s) \quad (2.6)$$

While it is possible that there exist more than one optimal policy, we denote all of them by π^* . As a final note, we observe how there always exists an optimal *deterministic* policy (Puterman, 2014), i.e., a policy for which in any state of the MDP there is only one action with a non-zero probability of being executed.

Aside from achieving the optimal state-value function, optimal policies achieve also the optimal *action-value function* q^* :

$$q^*(s, a) := \max_{\pi} q_{\pi}(s, a) \quad (2.7)$$

As we have noted before, both v^* and q^* must satisfy the Bellman equations. In their case, as they are optimal, their equation can be rewritten so that they do not refer any policy in particular. In fact, for the optimal state-value function, the value of a state is exactly the expected return of the action(s) that maximizes the expected return itself:

$$v^*(s) := \max_{a \in A} \sum_{s' \in S} T(s' | s, a) [R(s, a, s') + \gamma v^*(s')] \quad (2.8)$$

The optimal value of an action is instead equal to the expected immediate reward obtained by taking such action, plus the discounted value of the action maximizing the action-value function from the next state:

$$q^*(s, a) := \sum_{s' \in S} T(s' | s, a) \left[R(s, a, s') + \gamma \max_{a' \in A} q^*(s', a') \right] \quad (2.9)$$

Equations (2.8) and (2.9) are usually referred to as the “*Bellman optimality equations*”. As we will soon see, they play a crucial role in reinforcement learning algorithms.

2.2 Reinforcement Learning Algorithms

As we have stated, the goal in reinforcement learning is to *learn* a policy that allows the agent to achieve a task in an environment. This is done by using the experience samples that the agent gathers during the training episodes to update the agent’s current policy. The main question is then: *how* should the agent update its policy? Learning algorithms provide an answer to this question.

All RL algorithms must, in a sense, solve two different problems. The first, and most obvious one, is often referred to as the *control problem*. The control problem consists in finding an optimal policy that the agent can use to achieve the task. As

such, the control problem revolves around learning a way to *control* the agent so that it can achieve the task in an optimal manner. To solve the control problem, RL algorithms update the policy they have learned so far by trying to further improve them (with respect to the accrued expected rewards). However, how can an RL algorithm decide whether the policy it has learned is optimal or not? To address this, they can solve the *prediction problem*. The prediction problem consists in evaluating policy and actions, usually by assigning them some numerical value. Solving the prediction problem thus requires RL algorithms to explore the environment to test and evaluate the policy it has currently learned. While the prediction problem seems to provide a less direct approach to learn how to achieve the task compared to the control problem, solving it is crucial in some RL algorithms as we will soon see.

There are several ways that RL algorithms try to tackle the problems we have just introduced. They can, for example, try to learn an approximation of the MDP in which the agent acts, and then generate a policy from it. Or, they can try to approximate the state- and action-value functions via neural networks, instead of saving the values in a table. In what follows, we aim at providing a set of basic ways of characterizing RL algorithms. We note that we will not present all the possible characterizations, but only those relevant to this dissertation.

RL algorithms can be classified using two main criteria. The first concerns whether the algorithm uses the learned policy or not to generate new experience samples (*on-policy* or *off-policy*). The second whether the algorithm learns or not a model of the MDP in which the agent acts to produce a policy (*model-based* or *model-free*). Following, we shortly introduce how algorithms can be categorised according to each criterion, and provide advantages and disadvantages for each category.

On-Policy and Off-Policy Algorithms The second aspect we use to categorise RL algorithms is whether the policy that is used to generate the experiences is the same as the one that is being optimised or not. In a sense, we can see the agent having two different policies: a *target* policy and a *behavioural* policy. The behavioural policy is used to generate experiences (and it can be derived from the current target policy or not), whereas the target policy is the one that the algorithm is optimising. The RL algorithms in which the target and behaviour policies are different referred to as *off-policy*, since the policy they use to generate the experiences is different from the one they are optimising. On the other hand, if the two policies are the same, then the algorithm is *on-policy*.

Off-policy algorithms can result in faster learning, as they can re-use samples from previous training episodes by storing them in a buffer which is used whenever the policy is updated. However, they tend to suffer from more *unstable* learning as they can overestimate the value of some actions, especially if the behavioural policy does not explore enough the action space. On the other hand, on-policy algorithms are more stable as the learning is based only on samples from one policy. This is crucial in continuous domains, where instability occurs naturally due to the size of the state and action spaces and the stochasticity of the environment. However, this comes at the cost of slower convergence speed.

Model-Free and Model-Based Algorithms The first aspect we consider is whether the learning algorithm tries to learn an approximation of the underlying MDP or not. In the case of *model-based* algorithms (Moerland et al., 2023), the agent not only learns a policy but also tries to learn the dynamics of the MDP. Then, the agent can compute a policy by using planning techniques over the learned MDP. On the other hand, in *model-free* algorithms the agent tries to directly learn a policy from the experiences it gathered during training and does not try to learn an approximation of the MDP. We note that in the rest of this thesis we will only use model-free algorithms.

Model-based algorithms tend to require less experience samples from the real MDP to produce good policies, as they can also use the learned MDP to produce new ones. Moreover, by using a model of the MDP they can generate policies that respect specific constraints by incorporating these in the planning phase. However, because of the planning phase, they are usually more expensive from a computational perspective. Model-based algorithms also tend to struggle in highly dimensional and stochastic domains, as in these cases learning an approximation of the MDP is challenging and requires a thorough exploration of the domain. On the other hand, as model-free algorithms are usually simpler and do not perform expensive operations on the MDP, they can easily be used in complex domains. As they do not learn a model, they also do not suffer from possibly biased approximations. However, compared to model-based approaches, they tend to require larger amounts of experience samples to learn good policies.

A large family of model-free RL algorithms can be classified as *temporal difference* algorithms. Amongst these we have Q-learning, an algorithm which is known to converge to an optimal policy, and which we will introduce more in-depth later. Given the importance of Q-learning in this dissertation, we give here a short presentation of temporal difference algorithms.

Temporal Difference Algorithms Temporal difference (TD) algorithms combine Monte Carlo algorithms (Kalos and Whitlock, 2009; A. Barto and Duff, 1994) and dynamic programming (Watkins, 1989; Kumar and Kanal, 1988). Monte Carlo algorithms are all those that wait for full episodes to end, and then use the real returns obtained from the episodes to update the state- and action-value functions and the policy. Their advantage is that they provide unbiased return estimates (as they use the real returns that the agent obtained), and that they can better tolerate long-horizon returns, as they consider only full episodes. On the other hand, they tend to be slower as they need entire episodes to perform updates, and the returns they use can have a high variance, as they depend on entire traces (and thus inherit the stochasticity of episodes). Dynamic programming algorithms require the MDP to be fully-known, and are known to converge to optimal policies. A well-known dynamic programming algorithm is value iteration.

Temporal difference algorithms draw from Monte Carlo algorithms by learning from experience samples obtained during training (thus not requiring a model of the environment), and from dynamic programming ones by updating estimates at each timestep by using the other estimates it has learned so far (thus not waiting for a complete episode).

TD algorithms solve the prediction problem via *bootstrapping*, i.e., by using the current estimates for the state-value function to update the state-value function itself. The simplest TD algorithm, TD(0), exploits the Bellman optimality equation by updating the state-value function of some policy π after each timestep t as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (2.10)$$

In Eq. (2.10), the state value function of the current state s_t is updated by adding to the current value of s_t the so-called *TD error*, i.e., the value $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, multiplied by the *learning rate* α . The learning rate α is a value between 0 and 1 which can be modified as the number of timesteps increases. The closer α is to 1, the larger the update to value of state s_t is, conversely, the closer it is to 0, the smaller the update to value of state s_t is.

TD(0) is known to converge (in the limit) to the real state-value function v_π for policy π (Tsitsiklis and Van Roy, 1997) under three conditions: (i) that actions are sampled according to the policy π , (ii) that each state-action pair is visited infinitely often, and (iii), that the learning rate decreases over time in such a way that the sum of learning rates diverges ($\sum_{t=0}^{\infty} \lambda_t = \infty$) but the sum of their squares converges ($\sum_{t=0}^{\infty} \lambda_t^2 < \infty$).

In RL, domains (thus MDPs) can be either *tabular* or not. A tabular domain is one where both the action and state spaces are reasonably small or, at least, finite. In the tabular case, the state- and action-value functions can both be represented as a table. However, whenever the size of at least one of the two spaces is too large or infinite, the state- and action-value functions should be represented by approximating the function with, usually, a neural network. In this case, the neural network encodes the state- and action-value functions performing what is called *function approximation*. Further, as this approach, and all those that stemmed from it, uses neural network, it can be considered a *neural RL* approach. In what follows, we will introduce several ways of classifying neural RL algorithms. We first introduce policy gradient algorithms. These are algorithms which directly modify the policy instead of trying to learn and optimise the state- and/or action-value functions. After, we introduce actor-critic algorithms. In these algorithms two networks are trained at the same time: an *actor* network, which determines how the agent acts (in practice, the actor is the policy), and a *critic* network, which is instead tasked with evaluating the actor's performance (in practice, the critic represents the state- and/or action-value functions, depending on the algorithm). Finally, we present trust-region algorithms. These algorithms have been introduced recently and are based on the idea to not modify policies too drastically with each update. PPO, which we will see later, is a trust-region algorithm.

Policy Gradient Algorithms In what follows, we assume that tasks are episodic and that there is no discounting, i.e., $\gamma = 1$ (although it is possible to use discounting in practice). Instead of trying to optimise the action-value function and derive a policy from it, policy gradient algorithms modify directly the policy itself by searching over the policy space. The theorem supporting policy gradient algorithms is the *policy gradient theorem* (Sutton, McAllester et al., 1999), which details how to compute the

gradient $\nabla_{\theta} v_{\pi_{\theta}}(s)$ for any MDP state. Without loss of generality, we denote with $J(\theta)$ the state-value of some (fixed) initial state s_0 given the current policy π_{θ} . The main result of the theorem is the following:

$$\nabla_{\theta} J(\theta) \propto \mathbb{E}_{\pi_{\theta}} \left[\sum_{a \in A} (q_{\pi_{\theta}}(s, a) \nabla_{\theta} \ln(\pi_{\theta}(a | s))) \right] \quad (2.11)$$

stating that the gradient with respect to the parameters θ of the state-value function is proportional to ($\propto$) the sum, across all MDP actions, of the expected value of the product between the gradient $\nabla_{\theta} \pi_{\theta}(a | s)$ and the action value $q_{\pi_{\theta}}(s, a)$. The expected value is calculated with respect to the probability of being in each state s when following the policy π_{θ} .

In practice, equation Eq. (2.11) is modified in two ways. First, a baseline function b is used to decrease the variance when estimating the gradient of the state-value function. Note that b must be chosen so that the expectation's value is not changed: this can be easily done by choosing the baseline function b to be a function depending solely on the MDP state, thus ignoring the action. A typical choice is thus to use the current estimate of the state-value function. As b satisfies this condition, we obtain that it does not modify the expected value, i.e., $\sum_{a \in A} b(s) \nabla_{\theta} \ln(\pi_{\theta}(a | s)) = 0$.

Secondly, note how in the right-hand side of Eq. (2.11) there is a sum across all MDP actions. As in practice this is infeasible (consider, e.g., a domain with a continuous action space), this sum is usually swapped with the return obtained at timestep t where the MDP's state is s_t and the agent takes some action a_t . This results in a Monte Carlo-style approach, where before updating the set of parameters θ the agent must have completed a training episode (of length h) so that we can calculate all the returns G_t for all timesteps $t = 0, \dots, h$. Thus, we obtain the following equation:

$$\nabla_{\theta} J(\theta) \propto \mathbb{E}_{\pi_{\theta}} [(G_t - b(s_t)) \nabla_{\theta} \ln(\pi_{\theta}(a_t | s_t))] \quad (2.12)$$

Equation (2.12) is the foundation of REINFORCE (modified with the baseline function) (Williams, 1992), the first policy gradient algorithm. From Eq. (2.12) it is then possible to derive the update rule for θ in REINFORCE:

$$\theta \leftarrow \theta + \alpha (G_t - b(s_t)) \nabla_{\theta} \ln(\pi_{\theta}(a_t | s_t)) \quad (2.13)$$

Actor-Critic Algorithms Actor-critic algorithms combine RL algorithms which directly manipulate policies (such as policy gradient algorithms) with algorithms which are based on learning state and/or action-value functions and deriving a policy from these. This family of algorithms is thus very diverse, as there are no other constraints on their definition aside from the one we have just given. Thus, in what follows, we will only provide a very high-level description of algorithms in this class.

As the name suggests, actor-critic algorithms have two components, an *actor* and a *critic*. The actor is a policy function π , and provides ultimately the policy learned by the algorithm. The critic is instead a value function, either in the form of a state-value function or of an action-value function. The objective of the critic is to estimate a specific value function, which is used to update the critic.

The actor and the critic can be trained with various algorithms. The actor can be trained with any policy gradient algorithm that uses an estimate of the state- and/or action-value functions, such as REINFORCE modified to use the state-value function as the baseline in Equation (2.12). whereas the critic with any algorithm that learns a value function, such as SARSA or TD(0). Crucially, the value function learned by the critic is used to update the actor, and the samples gathered by the actor are used to update the critic.

Trust-Region Algorithms Policy gradient algorithms can, in practice, be too unstable. This means that they can modify the policy in a too “aggressive” manner, moving from relatively good policies to worse-than-random ones in a handful of updates. Motivated by this issue, Schulman, Levine et al. (2015) proposed an algorithm called *trust region policy optimization* (TRPO) with guarantees that it monotonically improves policies, i.e., if TRPO updates policy π to policy π' , then $\pi' \geq \pi$. This algorithm gave then rise to a series of approaches with similar guarantees, which are referred to as *trust-region algorithms*. PPO, which we will introduce later, is one of such algorithms. Here, we will quickly outline the theoretical background of TRPO and the subsequent trust-region algorithms. However, before doing so, we introduce the necessary notation. We denote with A_π the *advantage function* which, given an MDP state s and action a , returns the difference between the action-value of a and the state-value of s , i.e., $A_\pi(s, a) := q_\pi(s, a) - v_\pi(s)$. Then, we denote with $\eta(\pi)$ the expected return of policy π , i.e., $\eta(\pi) := \mathbb{E}_{s_0 \sim \mu} [v_\pi(s_0)]$. Finally, we denote with $\rho_\pi(s)$ the discounted visitation probability of s while following policy π , i.e., $\rho_\pi(s) := \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s)$, where $\Pr(s_0 = s) = \mu(s)$ and $\Pr(s_t = s)$ is the probability the the MDP state at timestep t is s when following policy π .

The main characteristic of trust-region algorithms is that the policy updates they perform ensure that the updated policy is at least as good as the original policy. The equation at the base of trust-region algorithms was proven in (Kakade and Langford, 2002), and relates the expected return of some policy π' to the expected return and advantage function of a starting policy π :

$$\eta(\pi') = \eta(\pi) + \sum_{s \in S} \rho_{\pi'}(s) \sum_{a \in A} \pi'(a | s) A_\pi(s, a) \quad (2.14)$$

Equation (2.14) establishes that the expected return of policy π' can be derived by taking the expected return of policy π and modifying it by summing, for each MDP state s , the product between the probability of the MDP being in state s when following π' and the sum, across all MDP actions, of the product between the probability of taking action a from state s when following π' and the advantage of π for taking action a in state s .

The issue in Eq. (2.14) is given by the fact that, when we are updating policy π , we do not know to what π' it will get updated. It is therefore impossible to evaluate $\rho(\pi')$. However, assuming that π and π' are similar enough, it can be reasonable to estimate $\rho(\pi')$ by using $\rho(\pi)$. We thus can define an estimate $L_\pi(\pi')$ of $\eta(\pi)$ as follows:

$$L_\pi(\pi') := \eta(\pi) + \sum_{s \in S} \rho_\pi(s) \sum_{a \in A} \pi'(a | s) A_\pi(s, a) \quad (2.15)$$

As we have mentioned, if π' and π are close enough, then $L_\pi(\pi')$ can be a good estimate of $\eta(\pi)$. From this observation, Schulman, Levine et al. provide a theorem establishing the following inequality¹ between $\eta(\pi')$ and $L_\pi(\pi')$:

$$\eta(\pi') \geq L_\pi(\pi') - CD_{\text{KL}}^{\max}(\pi, \pi') \quad (2.16)$$

where $C = \frac{4 \max_{s,a} |A_\pi(s,a)|\gamma}{(1-\gamma)^2}$ and $D_{\text{KL}}^{\max}(\pi, \pi') := \max_s D_{\text{KL}}(\pi(\cdot | s) \parallel \pi'(\cdot | s))$ is the maximum, across all MDP states, KL divergence between the action distributions of π and π' . Indeed, in Eq. (2.16) it can be seen how the estimated return $L_\pi(\pi')$ is penalized by a quantity which is proportional to the maximum KL divergence between π and π' .

As the right-hand side of Eq. (2.16) provides a lower bound for the expected return of the new policy π' , the objective in trust-region algorithms is to maximize it so that the updated policy has a higher expected return. At each update, by choosing π' to be the one maximizing Eq. (2.16), we ensure that we obtain a monotonically improving sequence of policies $\pi_0 \leq \pi_1 \leq \pi_2 \leq \dots$.

Maximizing the right-hand side of Eq. (2.16) can be rewritten by using the definition of L as follows:

$$\underset{\pi'}{\text{maximize}} \left[\eta(\pi) + \sum_{s \in S} \rho_\pi(s) \sum_{a \in A} \pi'(a | s) A_\pi(s, a) - CD_{\text{KL}}^{\max}(\pi, \pi') \right] \quad (2.17)$$

We make a few observations. First, $\eta(\pi)$ can be ignored, as it is a constant value independent of π' . Second, $\sum_{s \in S} \rho_\pi(s)$ is equal to $\frac{1}{(1-\gamma)} \mathbb{E}_{s \sim \rho_\pi} [\dots]$. As $\frac{1}{(1-\gamma)}$ is a constant value, we can ignore it in the final optimization problem. Finally, $\sum_{a \in A} \pi'(a | s) A_\pi(s, a)$ is rewritten by including an importance sampling estimator that is used to account for selecting some action a_t according to policy π instead of π' :

$$\sum_{a \in A} \pi'(a | s) A_\pi(s, a) = \mathbb{E}_{a_t \sim \pi(\cdot | s_t)} \left[\frac{\pi'(a_t | s_t)}{\pi(a_t | s_t)} A_\pi(s_t, a_t) \right]$$

At this point, in (Schulman, Levine et al., 2015) it is noted that due to the theoretical value recommended for C , the policy updates would change policies in very small steps. Thus, they propose to constraint an expectation of the KL divergence to be below some threshold k . This is also the reason why these algorithms are called *trust region*, as this constraint enforces the updated policy to belong to a small region surrounding the original policy. We can then rewrite Eq. (2.17) as follows:

$$\underset{\pi'}{\text{maximize}} \mathbb{E}_{s_t \sim \rho_\pi, a_t \sim \pi(\cdot | s_t)} \left[\frac{\pi'(a_t | s_t)}{\pi(a_t | s_t)} A_\pi(s_t, a_t) \right] \quad (2.18)$$

$$\text{subject to } \mathbb{E}_{s_t \sim \rho_\pi} [D_{\text{KL}}(\pi(\cdot | s) \parallel \pi'(\cdot | s))] \leq k \quad (2.19)$$

Later, in (Schulman, Wolski et al., 2017) it is observed how the choice of using a constraint on the KL divergence (Eq. (2.19)) is, in truth, not technically motivated. A

¹We note that the inequality presented in Eq. (2.16) is given in the arXiv version of (Schulman, Levine et al., 2015), and not in the conference version of the paper.

new maximization problem is proposed by incorporating back the KL divergence in the expected value of Eq. (2.18) and scaling it by some coefficient β :

$$\underset{\pi'}{\text{maximize}} \mathbb{E}_{s_t \sim \rho_\pi, a_t \sim \pi(\cdot | s_t)} \left[\frac{\pi'(a_t | s_t)}{\pi(a_t | s_t)} A_\pi(s_t, a_t) - \beta D_{\text{KL}}(\pi(\cdot | s_t) \parallel \pi'(\cdot | s_t)) \right] \quad (2.20)$$

There are two issues in Equation (2.20): (i) evaluating the expected KL divergence is infeasible in large MDPs, and (ii), choosing a β that performs well even for a single problem is difficult.² PPO was then proposed in (Schulman, Wolski et al., 2017) as a way to specifically address these two issues.

We will now introduce two algorithms that will appear throughout this thesis: Q-learning (Watkins and Dayan, 1992) and Proximal Policy Optimization (PPO) (Schulman, Wolski et al., 2017). Q-learning is a classic RL algorithm which in tabular scenarios can provably converge to the optimal policy. PPO is, on the other hand, an algorithm specifically designed to train RL agents in domains with infinite state and action spaces. In the rest of the thesis we will see other RL algorithms, however here we introduce only these two as they are the only ones to appear in multiple of the following chapters.

2.2.1 Q-learning

Q-learning (Watkins and Dayan, 1992) is a *model-free, off-policy, temporal difference* algorithm which is used in *tabular* domains. A domain is tabular whenever its state and action spaces are finite: in such cases, it is possible to represent both the state- and action-value functions using tables.

Q-learning can be easily derived from TD(0) by making two modifications. First, as we have noted TD(0) is used to solve the prediction problem. Instead, we use Q-learning to solve the control problem: therefore, we use and update the action-value function instead of the state-value function. At this point it is worth noting that in Eq. (2.10) the estimate of the current state's value is updated by using the estimate of the next state's value. As in Q-learning we update the action-value function, we update the value of each state-action pair $\langle s_t, a_t \rangle$ by looking at a possible next state-action pair $\langle s_{t+1}, a' \rangle$. The second modification made in Q-learning is to set a' as the action that maximizes $Q(s_{t+1}, a')$. We thus obtain the update for the action-value function in Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \quad (2.21)$$

As the action a' taken at timestep $t + 1$ is chosen following a greedy policy with respect to the current estimated action-value function Q , the samples we use to update the target policy are not fully generated by the target policy itself. Note that the target policy cannot be greedy because otherwise it would not visit each state-action pair infinitely thus violating a crucial condition for convergence of Q-learning. Thus,

²Schulman, Wolski et al. (2017) note that this was the main reason behind the use of the hard constraint of Eq. (2.19) in the optimization problem in (Schulman, Levine et al., 2015).

Q-learning is an off-policy algorithm. By using the real action a_{t+1} , chosen according to the target policy, that the agent performs at the following timestep $Q(s_t, a_t)$, we instead obtain SARSA (Rummery and Niranjan, 1994), an on-policy control algorithm based on TD(0).

Q-learning is known to converge (in the limit) to an optimal policy under certain assumptions: (i) each state-action pair is visited infinitely often, and (ii), the learning rate decreases over time in such a way that the sum of learning rates diverges ($\sum_{t=0}^{\infty} \gamma_t = \infty$) but the sum of their squares converges ($\sum_{t=0}^{\infty} \gamma_t^2 < \infty$).

2.2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) (Schulman, Wolski et al., 2017) is a *model free, on-policy* algorithm used in *continuous (non-tabular)* domains. Continuous domains are all those where at least one between the state space and action space is infinite. This implies that, unlike tabular domains, action-value function is not representable with a table³. In these cases it is therefore necessary to use *function approximation* by, e.g., training a neural network to approximate the action-value function. In what follows, we denote with θ a set of parameters defining a differentiable policy π_θ . When using function approximation, it is crucial to update the parameters θ to improve the state-value function v_{π_θ} . This update is done by following the gradient $\nabla_\theta v_{\pi_\theta}$, i.e., the vector which, for each component of θ , indicates the direction and rate of fastest increase of the state-value function with respect to such component of θ .

PPO tackles the problems we have introduced when discussing trust-region algorithms with a relatively simple, but unexpectedly effective, change: replacing the KL-divergence addend in Eq. (2.20) with a clip function so that the updated policy will still be relatively close to the original one.

To declutter our notation in Eq. (2.20), we introduce the following abbreviations:

$$r_t := \frac{\pi'(a_t | s_t)}{\pi(a_t | s_t)} \quad A_t := A_\pi(s_t, a_t) \quad \mathbb{E}_t[\cdot] := \mathbb{E}_{s_t \sim \rho_\pi, a_t \sim \pi(\cdot | s_t)}[\cdot]$$

By removing the KL-divergence addend and using our new notation in Eq. (2.20), we obtain the following:

$$\underset{\pi'}{\text{maximize}} \mathbb{E}_t[r_t A_t] \quad (2.22)$$

As one can imagine, the problem in optimizing Eq. (2.22) is the absence of the KL-divergence addend, which makes the new policy π' drastically different from the original π . Indeed, observe how the ratio r_t is maximized whenever the denominator $\pi(a_t | s_t)$ is low and the numerator $\pi'(a_t | s_t)$ is high. In practice, the updated policy will try to maximize the probability of taking any action a_t at timestep t whenever the probability $\pi(a_t | s_t)$ is very low and the advantage A_t is high (> 0), as those terms have the greatest impact on the maximization objective in 2.22. On the other hand, whenever an action a_t at some timestep t is such that $\pi(a_t | s_t)$ is very high and the advantage A_t is low (< 0), the updated policy will try to minimize $\pi'(a_t | s_t)$.

³It should be noted that the state-value function can be represented as a table in case the state space is finite.

Thus, a clip function is introduced in Eq. (2.22) to avoid these extreme modifications to π , obtaining the following:

$$\underset{\pi'}{\text{maximize}} \mathbb{E}_t [\min (r_t A_t, \text{clip}(r_t, 1 - \epsilon, 1 + \epsilon) A_t)] \quad (2.23)$$

where ϵ is a hyperparameter called *clip parameter* or *clip ratio*, usually within 0.1 and 0.3.

The idea of using the clip function in Eq. (2.23) is to increase $\pi'(a_t | s_t)$ only until it reaches $(1 + \epsilon)\pi(a_t | s_t)$ whenever $A_t > 0$. Dually, whenever $A_t < 0$, we decrease $\pi'(a_t | s_t)$ only until it reaches $(1 - \epsilon)\pi(a_t | s_t)$. In this way, PPO can more effectively optimize the policy also for actions in other timesteps, and the updated policy π' is not too far from the original π . Moreover, PPO takes the minimum value between $r_t A_t$ and the clipped product, ensuring that its objective function is a lower bound of the unclipped objective $r_t A_t$ in Eq. (2.22).

In practice, PPO uses a neural network to encode a policy π and its state-value function v_π . As PPO uses a neural network, we re-introduce the θ notation for the rest of the subsection. Similarly to Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016), another actor-critic policy gradient algorithm, PPO gathers experience samples by running in parallel n copies of the agent acting in n different copies of the environment. After a certain number of timesteps T has elapsed, PPO updates the policy π_θ and its state-value function using the n sampled trajectories of length T . This is done by computing a gradient that modifies the set of parameters θ in order to maximize the following objective function $L_t^{CLIP+VF+S}$:

$$L_t^{CLIP+VF+S}(\theta) := \hat{\mathbb{E}}_t [L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \quad (2.24)$$

where c_1, c_2 are coefficients, and S denotes an entropy bonus, which previous works, e.g., (Williams, 1992; Mnih et al., 2016), suggest to use as it encourages the updated policy to not be too deterministic thus ensuring sufficient exploration. $\hat{\mathbb{E}}_t$ denotes instead the empirical expected value, evaluated from the n sampled trajectories. Indeed, note that Eq. (2.24) is computed for each of the n sampled trajectories. In other words, for each timestep $t \leq T$, PPO first computes the objective values of each of the sampled trajectories, and then updates the policy via gradient descent using the computed objectives.

Both $L_t^{CLIP}(\theta)$ and $L_t^{VF}(\theta)$ include an empirical target value for the timestep t , which is evaluated as follows from the gathered samples:

$$V_t^{\text{target}} := r_t + \gamma r_{t+1} + \dots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} v_\theta(s_T) \quad (2.25)$$

the objective function for the value function is then calculated as the squared-error loss between the learned state-value function v_θ and the target state-value function:

$$L_t^{VF} := \left(v_\theta(s_t) - V_t^{\text{target}} \right)^2 \quad (2.26)$$

From Eq. (2.25), we also derive an empirical value $\hat{A}_t$ for the advantage A_t :

$$\hat{A}_t := V_t^{\text{target}} - v_\theta(s_t) \quad (2.27)$$

the empirical advantage is then used in place of the true advantage function in L_t^{CLIP} , which is obtained from Eq. (2.23):

$$L_t^{CLIP} := \hat{\mathbb{E}}_t \left[\min \left(r_t \hat{A}_t, \text{clip}(r_t, 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \quad (2.28)$$

PPO was observed to perform extremely well in many complex scenarios, outperforming previous algorithms such as A3C and the original TRPO. Due to its on-policy nature, it tends to be data-hungry as after every update the gathered trajectories must be discarded. However, by incorporating a simpler but yet effective version of the trust-region approach we saw previously, it still manages to converge relatively fast.

2.3 Reward Machines

As we have seen, there are two standard assumption in reinforcement learning. First, both the environment in which the agent acts and the reward function (acting as a proxy for the task that the agent must learn to achieve) are *Markovian*. In other words, this means that, the dynamics of the environment and of the reward function depend solely on the current state, and not on the history so far. This is a critical assumption to ensure, for example, the convergence of several learning algorithms (e.g., Q-learning, SARSA). However, in many cases this assumption does not hold (Littman et al., 2017; Gaon and Brafman, 2020). Second, both the dynamics of the environment and the reward function are kept “hidden” to the agent. While it is understandable to treat the environment’s dynamics as a *black box*, it is less understandable in the case of the reward function. Indeed, in most of the cases, the reward function is even coded by the programmer, who therefore must have complete knowledge about it. As such, it can be reasonable to expose part of the reward function to the agent.

These two observations are the core ideas behind the proposal of *reward machines* in (Toro Icarte, Klassen, Valenzano and McIlraith, 2018; Toro Icarte, Klassen, Valenzano and McIlraith, 2022; Toro Icarte, 2022). Reward machines are finite state automata encoding non-Markovian reward functions: at each timestep, the reward machine is in some state and, upon observing a high-level event related to the transition that occurred in the MDP, it transitions to a new state and produces an output reward. High-level events can represent that certain propositional symbols have become true or false. For example, if the environment in which the agent is acting represents an office, a high-level event can be the fact that the agent has entered a specific room of the office.

Reward machines come in two variants, changing how the output function is defined. In the first, the reward machine outputs a reward function depending solely on its state at the current timestep, making it a *Moore machine*. In the second, the reward machine outputs a reward based on its starting state and the observed high-level event, thus making it a *Mealy machine*. The Mealy-variant reward machines have been termed “*simple reward machines*” in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022; Toro Icarte, 2022). Nevertheless, as is the case with Moore and Mealy machines, these variants are equivalent. As in both variants reward machines are finite state automata, they can encode any kind of non-Markovian reward function

where the non-Markovian dependency on the history can be encoded in a regular language (where the language's alphabet is the RM's alphabet). In this thesis, we will only use reward machines in the Mealy variant.

Definition 2.3 (Reward Machine). A reward machine (RM) is a tuple $\mathcal{R} = \langle U, u_0, F, \Sigma, \delta_u, \delta_r \rangle$, where U is a finite non-empty set of states, $u_0 \in U$ is the initial state, F is the set of final states such that $F \cap U = \emptyset$, Σ is the alphabet of input symbols, $\delta_u : U \times \Sigma \rightarrow U \cup F$ is the transition function, and $\delta_r : U \times \Sigma \rightarrow \mathbb{R}$ is the output reward function.

In most of the works of Toro Icarte, Klassen, Valenzano and McIlraith, reward machines were defined starting from a set of propositional symbols *Prop*. The powerset of *Prop* was then used as the input alphabet Σ for the RM. From this point of view, we use a definition which is less strict, allowing for high-level events to be any kind of symbol in a generic input alphabet Σ .

The execution of an RM always starts in state u_0 , and at each timestep it is in some state $u \in U \cup F$. As the agent acts in the MDP a *labelling function* $L : S \times A \times S \rightarrow \Sigma$ produces the high-level event that is used as the input for the RM. Assuming that in the starting MDP state s the agent performs action a and the MDP then transition to state s' , the labelling function returns the symbol $\sigma = L(s, a, s')$. The RM then updates its state from u to $u' = \delta_u(u, \sigma)$, and it gives in output the reward $r = \delta_r(u, \sigma)$.

Example 2.1. Consider the “OFFICEWORLD” domain in Fig. 2.1 from (Toro Icarte, Klassen, Valenzano and McIlraith, 2022). In this domain, which rendition is given in Fig. 2.1a, the agent (the purple triangle) has to navigate a gridworld to complete tasks such as visiting rooms (marked with letters), or collecting items (such as a cup of coffee or a letter) and bringing them to specific locations, all while avoiding decorations (the asterisk in the rendition).

Figure 2.1b shows a reward machine representing a task for this domain. The task encoded by the reward machine requires the agent to collect the cup of coffee and bring it to the office (marked by the “o”). While doing so, the agent must also avoid all decorations.

State u_0 is the initial state, and the black circles represent terminal states. Every edge between some state u and some (not necessarily different) state u' is labelled by a tuple $\langle \varphi, r \rangle$, where φ is a formula of classical propositional logic and r is the output reward. In this domain, the labelling function labels each triple $\langle s, a, s' \rangle$ with the symbol contained in s' (if there is no symbol in s' , then the label is a dummy symbol corresponding to nothing). For example, if the agent reaches a cell containing the ☕ symbol, the label given in output by the labelling function is $\sigma = \text{☕}$. In this scenario, we say that a “label *satisfies* a propositional logic formula” if and only if the label is consistent with the propositional logic formula. If the labelling function outputs some label σ such that σ satisfies φ , then the RM state is updated to u' and the RM outputs r . For example, suppose that the RM is in state u_0 . So long as the agent avoids the decorations $*$ and does not reach a cup of coffee ☕, the output label will be the dummy empty label. This dummy empty label is consistent only with the formula $\neg \text{☕} \wedge \neg *$, as neither of the propositional symbols ☕ and $*$ appear in the label. Hence, the RM will remain in state u_0 and output a reward of 0. When the agent

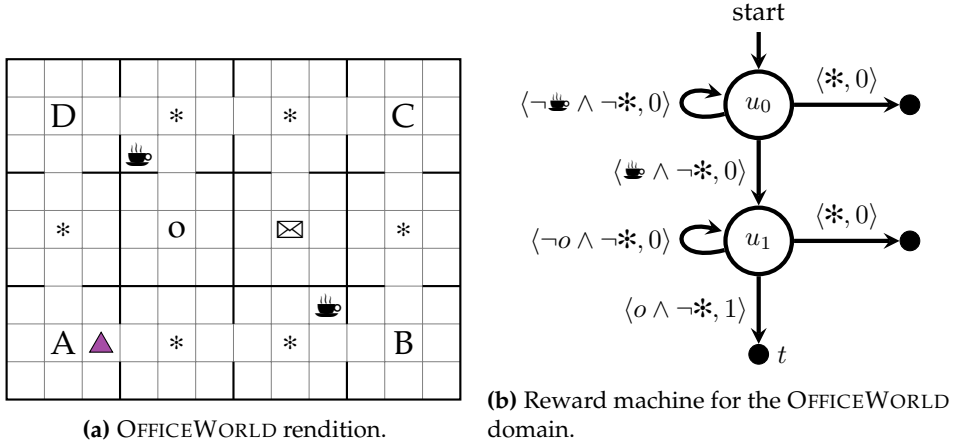


Figure 2.1: The OFFICEWORLD domain and a reward machine for it (both figures are from (Toro Icarte, Klassen, Valenzano and McIlraith, 2022)).

reaches a cell containing a ☕, we have that the label σ satisfies ☕ but not *, simply because $\sigma = \text{☕}$. Thus, the RM will transition from state u_0 to state u_1 , as the label on the edge between the two states is satisfied by σ , and it will output a reward of 0. Once the agent reaches the office cell o, the output label will be $\sigma = o$. The RM will then move from state u_1 to the final (success) state and output a reward of 1. If, at any point, the agent instead reaches a cell containing a *, the RM will instead transition from any state to a terminal (fail) state and output a reward of 0 as $\sigma = *$.

By combining a reward machine with an MDP (executing it in the way we have just outlined in the previous paragraph), we obtain a so-called “*Markov decision process with a reward machine*”, defined as follows.

Definition 2.4 (Markov Decision Process with a Reward Machine). A Markov decision process with a reward machine (MDPRM) is a tuple $\mathcal{T} = \langle S, \mu, S^F, A, T, \gamma, \Sigma, L, \mathcal{R} \rangle$, where $S, \mu, S^F, A, T, \gamma$ are as in an MDP, Σ is a finite set of events, $L : S \times A \times S \rightarrow \Sigma$ is a labelling function, and $\mathcal{R} = \langle U, u_0, F, \Sigma, \delta_u, \delta_r \rangle$ is a reward machine.

Given an MDPRM, a policy π of an agent trained in it is defined over the cross product between the set of states of the MDP and the set of states of the RM, i.e., $\pi : S \times U \rightarrow \Delta(A)$. It can be then observed that, in practice, an MDPRM induces a “*product*” MDP.

Observation 2.1. An MDPRM $\mathcal{T} = \langle S, \mu, S^F, A, T, \gamma, \Sigma, L, \mathcal{R} \rangle$ induces an MDP $M_{\mathcal{T}} = \langle S_{\mathcal{T}}, \mu_{\mathcal{T}}, S_{\mathcal{T}}^F, A_{\mathcal{T}}, T_{\mathcal{T}}, R_{\mathcal{T}}, \gamma_{\mathcal{T}} \rangle$, defined as follows:

- $M_{\mathcal{T}} := S \times (U \cup F)$, i.e., the set of states is the Cartesian product between the set of states of the MDP and the set of states, including the set of final states, of the RM;

- For arbitrary $s \in S$ and $u \in U$, we have that:

$$\mu_{\mathcal{T}}(\langle s, u \rangle) = \begin{cases} \mu(s) & \text{if } u = u_0 \\ 0 & \text{otherwise} \end{cases}$$

hence, the probability of some state $\langle s, u \rangle$ being the initial state is equal to the probability of s being initial whenever u is the initial RM state u_0 , and is 0 otherwise;

- $S_{\mathcal{T}}^F := \{\langle s, u \rangle \mid s \in S^F \vee u \in F\}$, i.e., the set of final states of the set of states in which at least one between the MDP state and the RM state is final;
- $A_{\mathcal{T}} := A$;
- For arbitrary $s \in S, u \in U, a \in A, s' \in S$, and $u' \in U \cup F$ we have that:

$$T_{\mathcal{T}}(\langle s', u' \rangle \mid \langle s, u \rangle, a) := \begin{cases} T(s' \mid s, a) & \text{if } \delta_u(u, L(s, a, s')) = u' \\ 0 & \text{otherwise} \end{cases}$$

thus, the probability of transitioning from $\langle s, u \rangle$ to $\langle s', u' \rangle$ via action a is equal to that of transitioning from s to s' via action a , under the condition that u' is the state to which the RM transitions from u upon observing $L(s, a, s')$. If the condition on the RM states is not satisfied, the transition probability is instead 0;

- For arbitrary $s \in S, u \in U, a \in A, s' \in S$, and $u' \in U \cup F$ with $\delta_u(u, L(s, a, s')) = u'$, we have that $R_{\mathcal{T}}(\langle s, u \rangle, a, \langle s', u' \rangle) := \delta_r(u, L(s, a, s'))$. The reward function is thus replaced by the output reward function of the reward machine;
- $\gamma_{\mathcal{T}} := \gamma$.

Having established a direct translation from MDPRMs to MDPs, any RL algorithm can be easily used to train an agent in the latter, hence training the agent in achieving the task encoded by the RM. At the same time, it is possible to exploit the structure of the RM to train agents more efficiently: this was, as we noted earlier, one of the main motivations behind the introduction of RMs. Following, we present how this can be done in several ways, as outlined in (Toro Icarte, 2022).

2.3.1 Exploiting the Structure of Reward Machines

As we mentioned previously, reward machines were introduced not only to encode non-Markovian reward functions, but also to expose (part of) the reward function to the agent. This can be done in several ways. In the following, we present the two that were used in the works of this thesis.

Counterfactual Experiences

The first way of exploiting the structure of reward machines is by generating *counterfactual* experiences and using them to augment policy training in off-policy RL

algorithms. We briefly outline how these counterfactual experiences are generated. Assume that the agent takes action a and the MDP transitions from state s to state s' . Then, we generate for each possible non-final state $\bar{u} \in U$ of the RM an experience sample assuming – *counterfactually* – that the RM’s current state is $\bar{u}$ and the input symbol is $L(s, a, s')$. In other words, for any non-final RM state $\bar{u} \in U$, we obtain from the RM the output reward $\delta_r(\bar{u}, L(s, a, s'))$ and the new counterfactual state $\delta_u(\bar{u}, L(s, a, s'))$. Thus, we generate the following set of experiences:

$$\{\langle s, \bar{u}, a, \delta_r(\bar{u}, L(s, a, s')), s', \delta_u(\bar{u}, L(s, a, s')) \rangle \mid \forall \bar{u} \in U\}$$

Note how the *real* current RM state u is included also in the set of experiences, as we consider all non-final RM states. After the set of experiences is generated, the RM is updated in the usual manner to the new *real* state $u' = \delta_u(u, L(s, a, s'))$.

This approach, called “*counterfactual experiences for reward machines*” (CRM), was first introduced in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022). CRM can be easily implemented in any off-policy algorithm, e.g., Q-learning, as can be seen in Algorithm 1.

Algorithm 1: Q-learning with counterfactual experiences for RMs (CRM)
(Toro Icarte, Klassen, Valenzano and McIlraith, 2022)

Input: MDPRM $\mathcal{T} = \langle S, \mu, S^F, A, T, \gamma, \Sigma, L, \mathcal{R} \rangle$, RM $\mathcal{R} = \langle U, u_0, F, \Sigma, \delta_u, \delta_r \rangle$,
number of training episodes num_episodes

```

1 For all  $s \in S, u \in U$ , and  $a \in A$ , initialize the value  $Q(\langle s, u \rangle, a)$  arbitrarily;
2 for  $\ell \leftarrow 0$  to num_episodes do
3   Initialize  $\langle s, u_0 \rangle \sim \mu$ ;
4   while  $s \notin S^F$  and  $u \notin F$  do
5     Choose action  $a$  from state  $\langle s, u \rangle$  using policy derived from  $Q$ ;
6     Take action  $a$  and observe the next state  $s'$ ;
7     Compute the reward  $r \leftarrow \delta_u(u)(s, a, s')$  and next RM state
        $u' \leftarrow \delta_u(u, L(s, a, s'))$ ;
8     Set  $Exp \leftarrow \{\langle s, \bar{u}, a, \delta_r(\bar{u}, L(s, a, s')), s', \delta_u(\bar{u}, L(s, a, s')) \rangle \mid \forall \bar{u} \in U\}$ ;
9     for  $\langle s, \bar{u}, a, r, s', \bar{u}' \rangle \in Exp$  do
10      if  $s' \in S^F$  or  $\bar{u}' \in F$  then
11         $Q(s, \bar{u}, a) \xleftarrow{\alpha} r$ ;
12      end
13      else
14         $Q(s, \bar{u}, a) \xleftarrow{\alpha} r + \gamma \max_{a' \in A} Q(s', \bar{u}', a')$ ;
15      end
16    end
17    Update  $\langle s, u \rangle \leftarrow \langle s', u' \rangle$ ;
18  end
19 end

```

A crucial result about CRM is the fact that, when coupled with Q-learning, which is guaranteed to converge when certain assumptions are satisfied (Watkins and Dayan,

1992), then the resulting learning algorithm will still be guaranteed to converge under the same assumptions. This is because CRM does not alter in any way how the real experiences are sampled, and just provides more samples per experience. As such, the same argument of (Watkins and Dayan, 1992) to prove convergence of Q-learning still applies to Q-learning when combined with CRM.

In (Toro Icarte, Klassen, Valenzano and McIlraith, 2018), another counterfactual-based algorithm called “*Q-learning for reward machines*” (QRM) was proposed. The main difference between CRM-augmented Q-learning and QRM, as noted in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022), is the fact that in the former the agent learns a single policy for all the RM states, whereas in the latter the agent learns a policy for each RM state. In tabular cases, there is in practice no difference between Q-learning augmented with CRM and QRM. On the other hand, when function approximation is required (e.g., because of an infinite state space) Q-learning augmented with CRM and QRM differ as the former needs to train only one network to approximate the value function, whereas the latter needs to train a network per each RM state. This makes the implementation of CRM for algorithms that use function approximation much easier than that of QRM.

Hierarchical Reinforcement Learning with Reward Machines

The second way of exploiting RM structure we present is grounded in the *options framework* (Sutton, Precup et al., 1999). The idea of the options framework is to use a *divide and conquer* approach to increase sample efficiency in reinforcement learning. Instead of trying to directly achieve the entire task, the agent learns to achieve sub-tasks, which are then combined in order to complete the entire task. An *option* thus represents a sub-task that the agent can perform while trying to achieve the task that is encoded by the reward function (or reward machine, in our case). For example, in the OFFICEWORLD domain, an option can consist in collecting the mail. The agent learns two kinds of policies when using this framework: one for each option, and a “*meta policy*”, the latter choosing the next option (amongst the possible ones) to achieve once one has been terminated. When the meta policy chooses some option as the next to achieve, the agent will start using the policy of such option to try to complete it.

In (Toro Icarte, Klassen, Valenzano and McIlraith, 2022), an options-based approach called “*hierarchical reinforcement learning for reward machines*” (HRM) was introduced. Fix an RM $\mathcal{R} = \langle u, u_0, F, \Sigma, \delta_u, \delta_r \rangle$. The main idea behind HRM is to treat each edge $\langle u, u_t \rangle$ in the reward machine as an option, and to have the meta policy decide to which state the RM should transition to. Note how in principle an option’s objective can also be to have the agent loop in the same RM state, if $\langle u, u \rangle$ is an edge in the RM. Thus, the set of options of the reward machine $\mathcal{R}$ is

$$\mathcal{A} = \{ \langle u, \delta_u(u, \sigma) \rangle \mid u \in U, \sigma \in \Sigma \}$$

When using this reward machine-based hierarchical approach, the meta policy is a policy of the form $\pi : S \times U \rightarrow \Delta(U)$, where $\pi(u_t \mid \langle s, u \rangle) \neq 0$ only if there is some symbol σ in the RM’s alphabet such that $\delta_u(u, \sigma) = u_t$. For each option $\langle u, u_t \rangle$, we define policies as functions $\pi_{\langle u, u_t \rangle} : S \rightarrow \Delta(\mathcal{A})$, i.e., a function mapping each state of

the MDP to a probability distribution over the set of MDP actions. The goal of the policy of option $\langle u, u_t \rangle$ is to move the RM state from u to u_t .

For each option $\langle u, u_t \rangle$ we must also define an *initiation set* $\mathcal{I}_{\langle u, u_t \rangle}$ and a *termination condition* $\beta_{\langle u, u_t \rangle}$.

The initiation set of an option is the set of states of the MDP from which the meta policy can choose the option as the next one to be achieved. In the case of HRM, the initiation set of some option $\langle u, u_t \rangle$ is any state in the product MDP in which the RM state is u , i.e.,

$$\mathcal{I}_{\langle u, u_t \rangle} := \{ \langle s, u \rangle \mid s \in S \}$$

The termination condition is a function that indicates, for each state of the MDP, whether the current option can be considered terminated or not in it (regardless of whether the agent managed to achieve or fail the option). In HRM, an option $\langle u, u_t \rangle$ terminates whenever either the new RM state u' is different from u (i.e., the RM state in which the option was started), or when the new MDP state s' is terminal.

As we noted, whenever an option is terminated it does not necessarily imply that the option was achieved. Indeed, for an option $\langle u, u_t \rangle$, if the new product MDP state $\langle s', u' \rangle$ is such that $u' \neq u_t$, then the option is actually failed. On the other hand, if $u' = u_t$, then the option is achieved. As such, two “special” reward hyperparameters r^+, r^- are used in HRM to train the options’ policies. These are used by modifying the reward function to train the option $\langle u, u_t \rangle$ as follows:

$$R_{\langle u, u_t \rangle}(s, a, s') := \begin{cases} \delta_r(u, L(s, a, s')) + r^+ & \text{if } u \neq u_t \text{ and } \delta_u(u, L(s, a, s')) = u_t \\ \delta_r(u, L(s, a, s')) + r^- & \text{if } u \neq u_t \text{ and } \delta_u(u, L(s, a, s')) \neq u_t \\ \delta_r(u, L(s, a, s')) & \text{otherwise} \end{cases} \quad (2.29)$$

Notice how the special rewards are used only when the option’s goal is not to loop in the same state u ; in that case, only the “normal” RM reward is used.

Algorithm 2 shows the pseudocode for combining HRM with Q-learning, as was done in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022; Toro Icarte, 2022). As it was presented, the algorithm included also counterfactual reasoning (lines 14-21). Aside from the “real” experience, the algorithm generates a synthetic experience per each possible option. Then, the synthetic experience is used to update its corresponding option policy, thus improving sample efficiency as in “vanilla” CRM.

Finally, we note that, unlike CRM, HRM does not guarantee convergence to optimal policies. This is because option-based approaches are “short-sighted”, meaning they prefer optimizing rewards obtained while trying to achieve each individual option but not while trying to achieve the entire task. For example, as can be seen in the rendition of OFFICEWORLD in Figure 2.1a, if the agent needs to grab the coffee, it is more convenient (from the starting position) to collect the coffee closer to the “B” location, as opposed to the one that is closer to the office (the cell labelled with “o”). However, if the final objective is to bring the coffee to the office, then it is more optimal to collect the coffee which is more distant from the starting position, as this one is much closer to the office and would thus require the agent to take less actions to achieve the task.

Algorithm 2: Hierarchical Q-learning for reward machines (HRM) (Toro Icarte, Klassen, Valenzano and McIlraith, 2022)

Input: MDPRM $\mathcal{T} = \langle S, \mu, S^F, A, T, \gamma, \Sigma, L, \mathcal{R} \rangle$, RM $\mathcal{R} = \langle U, u_0, F, \Sigma, \delta_u, \delta_r \rangle$,
 number of training episodes num_episodes

- 1 For all $u \in U$, set $\mathcal{A}(u) \leftarrow \{u_t \mid \exists \sigma \in \Sigma. \delta_u(u, \sigma) = u_t\}$;
- 2 For all $s \in S, u \in U$, and $u_t \in \mathcal{A}(u)$, initialize the (meta) value $Q(s, u, u_t)$ arbitrarily;
- 3 For all $s \in S, u \in U, u_t \in \mathcal{A}(u)$, and $a \in A$, initialize the (option) value $Q_{\langle u, u_t \rangle}(s, a)$ arbitrarily;
- 4 **for** $\ell \leftarrow 0$ **to** num_episodes **do**
- 5 Initialize $\langle s, u_0 \rangle \sim \mu$, and $u_t \leftarrow \emptyset$;
- 6 **while** $s \notin S^F$ **and** $u \notin F$ **do**
- 7 **if** $u_t = \emptyset$ **then**
- 8 Choose option $u_t \in \mathcal{A}(u)$ using policy derived from Q ;
- 9 Set $r_t \leftarrow 0$ and $t \leftarrow 0$;
- 10 **end**
- 11 Choose action a from s using policy derived from $Q_{\langle u, u_t \rangle}$;
- 12 Take action a and observe the next state s' ;
- 13 Compute the reward $r \leftarrow \delta_r(u, L(s, a, s'))$ and next RM state $u_t \leftarrow \delta_u(u, L(s, a, s'))$;
- 14 **for** $\bar{u} \in U, \bar{u}_t \in \mathcal{A}(\bar{u})$ **do**
- 15 **if** $s \in S^F$ **or** $\delta_u(\bar{u}, L(s, a, s')) \neq u$ **then**
- 16 $Q_{\langle \bar{u}, \bar{u}_t \rangle}(s, a) \xleftarrow{\alpha} R_{\langle \bar{u}, \bar{u}_t \rangle}(s, a, s')$;
- 17 **end**
- 18 **else**
- 19 $Q_{\langle \bar{u}, \bar{u}_t \rangle}(s, a) \xleftarrow{\alpha} R_{\langle \bar{u}, \bar{u}_t \rangle}(s, a, s') + \gamma \max_{a' \in A} Q_{\langle \bar{u}, \bar{u}_t \rangle}(s', a')$;
- 20 **end**
- 21 **end**
- 22 **if** $s' \in S^F$ **or** $u' \neq u$ **then**
- 23 **if** $s' \in S^F$ **or** $u' \in F$ **then**
- 24 $Q(s, u, u_t) \xleftarrow{\alpha} r_t + \gamma^t r$;
- 25 **end**
- 26 **else**
- 27 $Q(s, u, u_t) \xleftarrow{\alpha} r_t + \gamma^t r + \gamma^{t+1} \max_{u'_t \in \mathcal{A}(u')} Q(s', u', u'_t)$;
- 28 **end**
- 29 Set $u_t \leftarrow \emptyset$;
- 30 **end**
- 31 Update $s \leftarrow s'$ and $u \leftarrow u'$;
- 32 Update $r_t \leftarrow r_t + \gamma^t r$;
- 33 Update $t \leftarrow t + 1$;
- 34 **end**
- 35 **end**

2.4 Linear-time Temporal Logic

Linear-time Temporal Logic (LTL) (Pnueli, 1977) is a modal logic that extends classical propositional logic with *temporal modalities*. Temporal logics, including LTL, have been used in several works in the literature to monitor AI systems (Alechina, Dastani et al., 2014), or to reason about their actions and changes (Chittaro and Montanari, 2000). In the context of reinforcement learning, LTL formulas can be used to express properties about evolving systems such as MDPs, but also to define reward functions for non-Markovian tasks (Camacho, Toro Icarte et al., 2019). Given the fact that LTL is amongst the simplest temporal logics, and most importantly its relevance in the chapters of this thesis, we introduce the logic in this section, presenting its syntax, its semantics, and its connection to RL.

2.4.1 LTL: Syntax

Fix a finite set of propositional symbols $Prop$. LTL formulas are defined from the following Backus-Naur form:

$$\varphi ::= p \in Prop \mid \top \mid \neg \varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid X \varphi \mid \varphi_1 U \varphi_2 \quad (2.30)$$

In Eq. (2.30), $X \varphi$ and $\varphi_1 U \varphi_2$ are the *temporal formulas* and X and U are the *temporal modalities*. $X \varphi$ is read as “*neXt* φ ”, and $\varphi_1 U \varphi_2$ is read as “ φ_1 *Until* φ_2 ”.

There are two other “standard” temporal modalities in LTL, the “*Future*” F and “*Globally*” G . We did not include these in the syntax above as they can be syntactically defined from $\top$, $\neg$, and U as follows:

$$F \varphi := \top U \varphi \quad G \varphi := \neg F \neg \varphi \quad (2.31)$$

2.4.2 LTL: Semantics

We now present the semantics of LTL over infinite sequences of sets of propositional symbols⁴ $\tau \in (2^{Prop})^\omega$, where 2^{Prop} is the power set of the set of propositional symbols. Sequences τ are also called “*traces*” or “*paths*”. Before giving the semantics of LTL, we introduce the necessary notation. We use 0 as the index of the first set of propositional symbols (if there is any) in τ . For a given trace τ , we denote with $\tau[i]$ the i -th set in τ , and with $\tau[i \dots]$ the subsequence obtained by taking the suffix of τ starting from the i -th set. We will also refer to the i -th set as the “ *i -th timestep*” of τ .

Let $t \in \mathbb{N}$ be any natural number representing an arbitrary timestep/index in τ ; from the LTL syntax in Eq. (2.30), we define the semantics for LTL formulas as follows:

$$\begin{array}{ll} \tau, t \models p & \text{iff } p \in \tau[t] \\ \tau, t \models \top & \text{always} \\ \tau, t \models \neg \varphi & \text{iff } \tau, t \not\models \varphi \\ \tau, t \models \varphi_1 \wedge \varphi_2 & \text{iff } \tau, t \models \varphi_1 \text{ and } \tau, t \models \varphi_2 \end{array}$$

⁴LTL semantics can also be defined over other kinds of models (such as labelled transition systems). However, we only present the semantics of LTL over traces, as in this thesis we are interested only in these.

$$\begin{aligned} \tau, t \models \varphi_1 \vee \varphi_2 & \quad \text{iff } \tau, t \models \varphi_1 \text{ or } \tau, t \models \varphi_2 \\ \tau, t \models X \varphi & \quad \text{iff } \tau[t + 1 \dots] \models \varphi \end{aligned} \quad (2.32)$$

$$\begin{aligned} \tau, t \models \varphi_1 \cup \varphi_2 & \quad \text{iff } \tau[i \dots] \models \varphi_2 \text{ for some } i \geq t, \text{ and} \\ & \quad \tau[j \dots] \models \varphi_1 \text{ for all } t \leq j < i \end{aligned} \quad (2.33)$$

where we read $\tau, t \models \varphi$ as “ τ satisfies φ from timestep t onwards”. We write $\tau \models \varphi$ as a shorthand for $\tau, 0 \models \varphi$. We can intuitively explain the temporal modalities from their semantics’ definition as follows: $X \varphi$ means that φ is true from the following timestep $t + 1$ (Eq. (2.32)), and $\varphi_1 \cup \varphi_2$ means that φ_2 will be true at some timestep following t , and that φ_1 is true from timestep t up until the timestep that φ_2 becomes true (Eq. (2.33)).

In Section 2.4.1 we have syntactically defined the F and G modalities using the U modality. While the semantics of these two modalities can also be derived from the semantics of U, we provide here also a version of them that does not depend on the latter modality:

$$\tau, t \models F \varphi \quad \text{iff } \tau[i \dots] \models \varphi \text{ for some } i \geq t \quad (2.34)$$

$$\tau, t \models G \varphi \quad \text{iff } \tau[i \dots] \models \varphi \text{ for all } i \geq t \quad (2.35)$$

indeed, $F \varphi$ means that φ will be satisfied by τ at some point starting from timestep t , and $G \varphi$ that φ is always true in τ starting from timestep t .

2.4.3 LTL_f: LTL for Finite Traces

In the previous subsection we have given the semantics of LTL for infinite traces. In terms of applications for RL, LTL can thus be used for tasks without a finite horizon, i.e., *continuous tasks*. However, specifically for the same reason, LTL cannot be used in tasks with a finite horizon, i.e., *episodic tasks*. For the latter tasks, we can use LTL_f (De Giacomo and Vardi, 2013), a variant of LTL with semantics specifically defined for finite traces.⁵

The syntax of LTL_f syntax is exactly the same as vanilla LTL; as such, we will not re-introduce it. On the other hand, semantics are different. As one can imagine, models of LTL_f formulas are finite sequences $\tau \in (2^{Prop})^*$. For a finite sequence τ , we denote, intuitively, its length with $length(\tau)$. As in LTL, we index with 0 the first set of propositional symbols (if there is any) in τ , and with *last* the index of the last set. Note how $last = length(\tau) - 1$. Following, we give the semantics of LTL_f only for the temporal formulas, as the semantics for the propositional formulas are the same as for LTL:

$$\begin{aligned} \tau, t \models X \varphi & \quad \text{iff } t < last \text{ and } \tau, t + 1 \models \varphi \\ \tau, t \models \varphi_1 \cup \varphi_2 & \quad \text{iff } \tau[i \dots] \models \varphi_2 \text{ for some } last \geq i \geq t, \text{ and} \\ & \quad \tau[j \dots] \models \varphi_1 \text{ for all } t \leq j < i \\ \tau, t \models F \varphi & \quad \text{iff } \tau[i \dots] \models \varphi \text{ for some } last \geq i \geq t \\ \tau, t \models G \varphi & \quad \text{iff } \tau[i \dots] \models \varphi \text{ for all } last \geq i \geq t \end{aligned}$$

⁵This important point was brought to my attention by dr. Nima Motamed, arising from conversations he had with dr. Paniscete Sneiche.

As a concluding remark over LTL_f , consider the formula $Last := \neg X \top$. Note that, for any sequence τ , given the LTL_f semantics of X , $last$ is the only timestep for which $Last$ is true. $Last$ can thus be used to identify the last timestep in τ .

Relationship with Reinforcement Learning We conclude this section by connecting LTL to the main topic of this thesis, reinforcement learning. As we have mentioned in the opening paragraph of this section, we can use LTL to express properties on evolving systems, including MDPs. We note that, while the semantics we have provided for LTL are defined over infinite sequences, there exist also several variants for which the semantics are defined over finite traces. An example is the “finite” variant of LTL, LTL_f (De Giacomo and Vardi, 2013), or the “past” variant of LTL, Pure-Past LTL (PPLTL) (Zuck, 1986; De Giacomo, Di Stasio et al., 2020). Hence, we can also express objective for episodic tasks when using temporal logics.

As we have seen in the definition of reward machines, it is possible to “label” each state of the MDP if we use a labelling function. Such labelling function could, for example, label each MDP state with a set of propositional symbols (intuitively, those that are true in such state). There are thus several ways to employ LTL and its variants in reinforcement learning.

The simplest way of using LTL is to encode a reward function using formulas. If each MDP state is labelled with a set of propositional symbols, then each episode induces a sequence of sets of propositional symbols. It would then be possible to reward the agent in each episode (in Monte-Carlo fashion) depending on whether the formula is satisfied by the episode’s induced sequence or not. This is done in several works in the literature (Fu and Topcu, 2014; Littman et al., 2017; X. Li, Vasile et al., 2017; Hahn et al., 2019; M. Hasanbeig, Kantaros et al., 2019).

More relevant in this thesis is the idea of using LTL and its variants to synthesise reward machines or similar objects. Indeed, from formulas of LTL and some of its variants, like LTL_f , it is possible to synthesise finite state automata that recognise all and only the traces that satisfy the original formula (Pnueli, 1977; De Giacomo and Vardi, 2015). Generating a reward machine is then relatively easy because we only need to define the reward function. For example, Camacho, Toro Icarte et al. (2019) synthesise reward machines from several temporal logics, including LTL and LTL_f . De Giacomo, Iocchi et al. (2020) also synthesise restraining bolts, which are also finite state automata encoding non-Markovian reward functions, from temporal logics such as LTL_f . Finally, C. Zhu et al. (2024) show how to synthesise reward machines from GR(1) (Bloem, Jobstmann et al., 2012), a fragment of LTL.



3 Related Works

Before delving into the dissertation's contents, we first present areas of RL research that are closely related to this work.

We will first discuss the literature concerning reward machines. We will present works that make use of them in both single- and multi-agent reinforcement learning. Then, we will specifically focus on two lines of research concerning reward machines. We present works in which reward machines are automatically synthesised, instead of being defined by a human. By automatically synthesising reward machines, these works aim at: (i) reducing human-induced errors in the definition of reward machines, and (ii), providing techniques to easily obtain reward machines, regardless of the task's complexity. Then, we present works that extend reward machines by modifying either how they are defined or how they interact with the environment and the agent.

Following, we include a more general discussion of works that use declarative AI approaches or techniques from formal methods or logic in reinforcement learning. We make a distinction between two lines of work in this area. In one, we include works which goal is to improve the convergence speed and/or the quality of the policy obtained after training. In the other, we consider only works that are used in *safe* reinforcement learning, where the goal is to train agents in not only achieving tasks, but also while complying with safety constraints.

3.1 Reward Machines

As reward machines are the topic of several chapters of this thesis, in this section we survey their literature. We first focus on works set in single-agent RL, and then move onto those in multi-agent RL. Then, we present works that specifically synthesise reward machines via logic-based specifications, or using declarative techniques, or by learning them from experience. Finally, we include works that proposed extensions or modifications of reward machines.

3.1.1 Single-Agent Reinforcement Learning

Reward machines were first introduced in (Toro Icarte, Klassen, Valenzano and McIlraith, 2018), where they were used in single-agent RL. In the paper, aside from

introducing RMs, Toro Icarte, Klassen, Valenzano and McIlraith introduced also QRM, an off-policy algorithm that exploits the structure of RMs to increase sample efficiency by generating counterfactual experiences and which keeps a table for action values for each RM state. Later, Toro Icarte, Klassen, Valenzano and McIlraith expanded their work on RMs in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022), introducing CRM, a second algorithm that uses counterfactual experiences but which keeps a single table for action values instead of one per each RM state, and HRM, a hierarchical algorithm based on the options framework of Sutton, Precup et al. The CRM algorithm in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022) was defined as a modification of Q-learning; later, in (Sun and Lespérance, 2023), modified versions of the soft actor critic (SAC) (Haarnoja et al., 2018) and twin delayed DDPG (TD3) (Fujimoto et al., 2018) deep RL algorithms that use counterfactual RM-based experiences were also introduced.

In (Koprulu and Topcu, 2023), reward machines are used to aid the agent in automatically generating curricula (Bengio et al., 2009) to improve sample efficiency in training agents to achieve non-Markovian tasks.

Reward machines have also been used in partially observable RL. The first work to use reward machines in POMDPs was (Toro Icarte, Waldie et al., 2019), which was later extended in a journal version (Toro Icarte, Klassen, Valenzano, Castro et al., 2023). Recently, (A. C. Li et al., 2024) used reward machines in partially observable domains where the labelling function that outputs the input symbols for the RM can also be noisy.

Finally, reward machines have also been applied in several real-life domains. In (Komp et al., 2024), it is shown how to learn, from expert demonstrations, a reward machine that can be used to train an agent to synthesise a cardiac pacemaker. (Camacho, Varley et al., 2021) and (Baert et al., 2024) instead use reward machines in robotics, with the former providing also real-life demonstrations.

3.1.2 Multi-Agent Reinforcement Learning

The first work to use reward machines in multi-agent RL (MARL) was (Neary et al., 2021). Specifically, they assumed that all agents cooperate to achieve the common task which is encoded by the team's RM, thus setting the work in *cooperative* MARL. The idea of the approach is to split the team's RM into several RMs, one per agent, and then train the agents individually to achieve the task. Later, in (Smith et al., 2023), several ways to automatically decompose the team's RM are introduced. Similarly to Neary et al., in (Hu et al., 2024) an approach that trains teams of agents with RMs in a decentralised manner is introduced. Specifically, Hu et al. provide each agent with information pertaining only other agents that are within a certain distance. This choice is motivated by showing that the dependency of each agent's action values with respect to the information over another agent decreases exponentially as the distance between the agents increases.

Notably, in both works from Neary et al. and Smith et al. the team's RM is defined by hand. Motivated by this, in (Varricchione, Alechina, Dastani and Logan, 2023; Varricchione, Alechina, Dastani and Logan, 2026) we introduced an approach that uses alternating-time temporal logic (ATL) (Alur et al., 2002) to automatically synthesise

the team's RM. We will present this work in more detail in Chapter 5. Similarly to our work, in (C. Zhu et al., 2024) an approach to automatically synthesise RMs in cooperative MARL is also proposed. However, instead of using ATL, C. Zhu et al. use GR(1) (Bloem, Jobstmann et al., 2012), a fragment of LTL. We note that, as ATL and GR(1) are incomparable in terms of expressive power, RMs obtained using our approach and the one of C. Zhu et al. do not encode the same sets of tasks. Finally, in (Trapasso and A. Jonsson, 2025), RMs for MARL are synthesised from partial-order plans (POPs). Each agent is endowed with its RM which is obtained from the POP, and is then trained individually, as in the approach of (Neary et al., 2021).

3.1.3 Synthesising Reward Machines

Ever since their introduction, there have been several approaches that tried to synthesise reward machines using, for example, declarative specifications, or by learning them directly from the experience samples and trajectories collected during training. In this section, we present several of these works.

We first consider works where reward machines are obtained from declarative specifications, e.g., logical formulas or plans. The first work proposing such an approach was (Camacho, Toro Icarte et al., 2019), where reward machines are synthesised from formulas of various temporal logics, such as LTL and LTL_f . Other works that use logic to synthesise reward machines are (Varricchione, Alechina, Dastani and Logan, 2023) and (C. Zhu et al., 2024) (the former which we present in Chapter 5), both in a multi-agent setting. In (Illanes et al., 2019) an approach to synthesise reward machines given a sequential or a partial-order plan is proposed. These works later inspired (Varricchione, Alechina, Dastani and Logan, 2024) (which we present in Chapter 4), where reward machines are synthesised given the set of all partial-order plans that can achieve the task.

Another line of research focuses on learning reward machines from experience. In (Toro Icarte, Waldie et al., 2019) an approach is proposed that uses Tabu search to update the RM hypothesis, by searching through the trace data generated by the agent exploring the environment. (Xu et al., 2020) presented an approach where the RM hypothesis is updated whenever traces that are inconsistent with the current one are detected. In (M. Hasanbeig, Jeppu et al., 2021) RM-learning is reduced to SAT solving, and solved using DPLL. While these approaches do not require an abstract model of the environment in the form of a planning domain, they focus on learning a reward machine for a single task. In (Toro Icarte, Waldie et al., 2019) and (Xu et al., 2020) the agent learns a policy for each state of the RM hypothesis. However, when the latter is updated, it has to re-learn such policies from scratch ((Xu et al., 2020) tries to mitigate this issue by transferring a subset of the policies, but this is not always possible). Finally, (Ardon, Furelos-Blanco and Russo, 2023) proposes an approach to learn RMs, but in the context of cooperative multi-agent RL. Note that all these approaches assume that the agent is able to generate “positive” traces, i.e., traces in which the task is achieved, which in some domains can be difficult due to their complexity.

Finally, an emerging line of research leverages large language models to synthesise reward machines from natural language specifications. (Castanyer et al., 2025) and

(Alsadat, Gaglione et al., 2025) are examples of such works. (Bester et al., 2024) also propose to synthesise counting reward automata via natural language specifications.

3.1.4 Extending Reward Machines

In this section, we focus on works that extend the original definition of reward machines. Specifically, we consider works that either extend the expressive power of RMs (in terms of the set of non-Markovian tasks that they can encode), or that modify in some way the definition of vanilla RMs (without increasing the expressive power).

An extension of reward machines that equips them with counters, called “*counting reward automata*” (CRA), was proposed in (Bester et al., 2024). This extension is based on counter machines (Minsky, 1967; Fischer et al., 1968), finite state automata which, like CRAs, make use of counters storing non-negative integer values. Such counters can be incremented, decremented, or tested to check whether the value they store is 0. Following (Bester et al., 2024), (Varricchione, Klassen et al., 2025) proposed to augment reward machines by equipping them with a stack, hence defining *pushdown reward machines* - we present this work in Chapter 6 of this dissertation. Finally, (Donnelly et al., 2025) propose to synthesise reward machines using Runtime Monitoring Language (RML) (Ancona et al., 2021), a language that can be used to specify highly expressive properties (e.g., non context-free ones) in runtime verification. The resulting “RML Reward Machines” thus retain the increase in expressive power, and are able to recognise and reward non-regular behaviours.

The works we have just presented all propose extensions of RMs that allow to encode a larger set of non-Markovian reward functions compared to vanilla RMs. In the following paragraph, we focus instead on extensions of RMs that do not allow to encode more non-Markovian reward functions. (Corazza et al., 2022) proposed an extension of RMs where the output function returns a probability distribution over rewards, allowing their *stochastic* reward machines to model noise in the output reward. In (Umili, Argenziano, Barbin et al., 2023) and later (Umili, Argenziano and Capobianco, 2024), reward machines are instead modified to make both the output reward function and the state transition function probabilistic. Moreover, their visual/neural reward machines (introduced in the respective papers) accept any type of data in input, as opposed to vanilla RMs which can accept only symbolic input. Following, in (Levina et al., 2024) reward machines are modified to enable quantitative reasoning when processing the input events. Instead of requiring a symbolic input, transitions in their “*numeric reward machines*” can occur whenever, for example, the distance to reach a specific location in the MDP has decreased compared to the previous timestep. Moreover, their output reward function can also be defined using quantitative reasoning, by granting the agent a reward which depends on how distant it is from some location. In (Furelos-Blanco et al., 2023) reward machines are augmented by introducing a *hierarchy*. In such hierarchies, RMs are able to call other RMs during execution. However, as they assume that there is always a *leaf* RM that cannot call other RMs, and that each RM cannot be called by itself, even via recursive calls, these hierarchies are able to encode only the same non-Markovian reward functions that can be encoded by vanilla RMs. The same hierarchical variant of reward machines was then used in the context of cooperative multi-agent RL in

(Zheng and Yu, 2024). In the context of *continuous-time* MDPs (CTMDPs) (Guo and Hernández-Lerma, 2009), *continuous* RMs are introduced in (Falah et al., 2025) to adapt the RM formalism in CTMDPs. Finally, another modification of reward machines, *First-Order Reward Machines* (FORMs), is proposed in (Ardon, Furelos-Blanco, Parac et al., 2025), where the transitions of the RM are labelled by first-order logic formulas instead of just symbolic events.

3.2 Declarative Approaches for Sample Efficient RL

In an effort to improve sample efficiency and the quality of the resulting policy, several works in the RL literature use ideas and techniques from fields such as formal methods, logic, and declarative AI. By leveraging ideas from these areas, it is possible to embed expert knowledge to improve training quality, or to make the resulting behaviours more explainable. In this section, we will present part of these works. We will first consider papers that use logic, then move onto those that use planning, and finally present works that use automata.

We begin by presenting works that have used logic in RL. Most of these works guide the agent in achieving tasks by using temporal logics. The intuition is to have formulas encode the task that the agent should learn to achieve. Agents are then trained to learn a policy which objective is to maximise the probability of satisfying the temporal formula: as a result, the agent will then learn to achieve the task with a high probability. An example of this is (M. Hasanbeig, Kroening et al., 2022), where LTL formulas are translated to limit deterministic Büchi automata which are then used to train the agent. In (M. Hasanbeig, Kroening et al., 2020) the authors then extended the same framework to be used with deep RL algorithms.

Other works leverage the *compositionality* that LTL and other temporal logics allow when defining tasks. This is for example done in (Vaezipoor et al., 2021) and (Jothimurugan, Bansal et al., 2021). The approach proposed by Vaezipoor et al. trains agents in achieving the subtasks encoded by an LTL formula; in this way, the agent can also re-use the policy it has learned to achieve some subtasks if they appear in unseen tasks. On the other hand, Jothimurugan, Bansal et al. propose to train agents in achieving LTL-encoded tasks by first transforming these into graphs representing subtask/subgoals, train one policy per edge in the graph, and then use a Dijkstra-like algorithm to plan over the long horizon how to achieve the task.

As we have seen, reward machines were also proposed to easily define (a set of) non-Markovian reward functions. Non-Markovian reward have been the topic of several works in the RL literature, starting from (Bacchus et al., 1996) where decision processes with non-Markovian rewards (NMRDPs) were introduced. The literature on non-Markovian domains has been explored also in more recent works, e.g., (Brafman and De Giacomo, 2019; Brafman and De Giacomo, 2024) which introduced *regular decision processes* (RDPs), i.e., decision processes with non-Markovian rewards where the dependency on the past is *regular* (in the formal languages sense). As reward functions in RDPs have regular dependencies on the past, Brafman and De Giacomo leverage linear-time dynamic logic on finite traces (LDL_f) (De Giacomo and Vardi, 2013), a temporal logic which is as expressive as regular expressions, to encode such

functions.

In relational reinforcement learning (Džeroski, De Raedt and Blockeel, 1998; Džeroski, De Raedt and Driessens, 2001), states and actions are represented using a relational first-order language. This is then exploited by modifying Q-learning Q-RRL, an algorithm which stores the action-value function by using, instead of a table, a relational regression trees, which can store the action values in a more compact manner. Relational reinforcement learning was then extended to continuous environments and deep algorithms in (Zambaldi et al., 2018).

By now, several works in the RL literature leverage planning to improve training efficiency and to improve performance in tasks that require the agent to act over a long horizon. Planning has been combined with RL in several ways: for example, as we mentioned earlier in Chapter 2, planning is used in model-based algorithms, where the learned policy is obtained by deriving a plan from the learned model of the MDP. In this paragraph, we only consider works that use *symbolic* planning, i.e., a form of planning that uses a symbolic representation of the environment in which the agent acts, to guide the learning RL agent. A majority of works integrating symbolic planning with reinforcement learning makes use of options. For example, in (F. Yang et al., 2018) the actions in plans are options that the RL agent learns to achieve. As the agent learns policies for the options, their expected reward are used to update the planning component, which then produces (if there is) a new improved plan. In (Lyu et al., 2019) the approach proposed in (F. Yang et al., 2018) is then extended to deep RL. In (Illanes et al., 2020), symbolic planning is used to generate either sequential or partial-order plans made of options. Agents are also tested in being *taskable*, i.e., being able to achieve different tasks instead of just one. This is also possible thanks to planning and options: as the agent learns to achieve options instead of individual tasks, it can use the policy for any option in any plan in which such option appears. Finally, (J.-C. Liu et al., 2024) proposes to automatically infer subtasks from demonstrations. The subtasks are then used by a symbolic planner which outputs a plan that guides the agent in learning with deep RL algorithms.

Aside from reward machines, recently other works have also used automata-like structures in RL. In (De Giacomo, Iocchi et al., 2019) and (De Giacomo, Iocchi et al., 2020) *restraining bolts* are introduced. Restraining bolts are automata obtained from LTL_f or LDL_f specifications that can thus be used, for example, also in the context of RDPs. The *option machines* proposed in (Hengst et al., 2022) are instead grounded in the options framework, as the edges on the finite state automata are labelled exclusively by high-level actions. Inspired by reward machines and the options framework, (Han et al., 2024) proposes an approach where the agent learns a low-level policy and a high-level policy. The low-level policy is used to achieve subtasks, similarly to the option framework. The high-level policy is instead used to determine the order in which the subtasks should be achieved. Further, the high-level policy can be used to generate a tree-like non-Markovian reward structure, showing all possible ways of ordering the subtasks that the agent has learned. Finally, (Jothimurugan, Alur et al., 2019) gives an approach to synthesise automata from specifications in a formal language that allows to express reachability and safety tasks. Starting from a specification, they compile it into an automata and use it to shape rewards. The automata they obtain is also equipped with a numeric registers (and so are similar

to the counting reward automata from (Bester et al., 2024)), but the purpose of the registers is not to define more complicated temporal patterns but to keep track of the quantitative *degrees* to which subtasks have been completed and constraints satisfied (so as to provide shaped rewards).

3.3 Declarative Approaches for Safe RL

We conclude the chapter on related works by presenting approaches that are concerned with ensuring that the agent adheres to a set of safety constraints. We note that, like in the previous section, we will focus on works that employ techniques from formal methods and declarative AI. We will therefore not consider works that, for example, use constrained MDPs, e.g., (Q. Yang et al., 2023; Simão et al., 2021), or statistical methods, e.g., (Berkenkamp et al., 2017; Laroche et al., 2019). Several surveys have been written on safe RL; we refer the interested reader to the following ones: (García and Fernández, 2015), containing a survey of early safe RL algorithms, (Gu, L. Yang et al., 2024), presenting works following the survey of García and Fernández, (Wachi et al., 2024) presenting a survey of how safety constraints are formulated, and (Kushwaha et al., 2025) providing a survey of safe RL approaches for constrained MDPs.

In this section we will specifically focus on approaches that employ declarative techniques to guarantee the satisfaction of safety constraints, in so-called “*provably safe reinforcement learning*”. For further references, we refer the reader to (Krasowski, Thumm et al., 2023), which provides a comprehensive survey in the topic (including also works not based on declarative techniques).

One of the most prominent provably safe approaches in RL based on formal methods are shields, introduced in (Alshiekh et al., 2018) for single-agent RL and in (ElSayed-Aly et al., 2021) for multi-agent RL. A shield is an automaton, usually synthesised from LTL specifications representing the safety constraints, which is used to modify the action selected by the agent so that it can never violate the safety constraints: as such, shields are provably safe approaches. (Odriozola-Olalde et al., 2023) provides a survey of papers using shields in RL. Further, (Könighofer et al., 2025) provides a review on shields and on how to integrate them in RL. In Chapter 7 we present shields more in-depth, as they relate to the approach we present in the chapter.

Similarly to shields, (Wen et al., 2015) proposes an approach that requires the transformation of a Safety LTL formula into an automaton. They then synthesise a maximally permissive strategy using the underlying MDP (rather than an abstract MDP, as in shields). Their approach is therefore double exponential in the input formula in general. However, the complexity reduces to linear for a subset of Safety LTL formulas (formulas of the form $\varphi_0 \wedge \Box \varphi_1$ where φ_0 and φ_1 are Boolean combinations of atomic propositions, and formulas of the form $\bigcirc q$ where q is an atomic proposition).

In (X. Li, Serlin et al., 2019) control barrier functions (CBFs) are synthesised from LTL specifications. A CBF (Wieland and Allgöwer, 2007) is a set of constraints that, given the current MDP state, provide constraints on the actions so that the

chosen action is modified into the closest one which is safe with respect to the action constraints. As the action chosen by the agent is modified into another one, this approach is classified as an “action projection” approach in (Krasowski, Wang et al., 2020). Aside from synthesising a CBF from the LTL specification, X. Li, Serlin et al. also synthesise a reward function that is used to guide the agent; they then validate their approach by training a robotic arm to cook and serve a hot dog.

In (Mason et al., 2017) an approach that uses Probabilistic Computation Tree Logic (PCTL) (Hansson and B. Jonsson, 1994) to enforce safety is introduced. The authors first obtain an abstract MDP from the MDP in which the agent acts. In the abstract MDP, abstract policies that satisfy constraints expressed in PCTL are synthesised; the abstract policies are then used to constrain the set of actions the agent can perform when learning in the original MDP. If the abstract MDP correctly captures the dynamics of the underlying MDP, their approach guarantees that any abstract policy returned satisfies the constraints and is Pareto optimal with respect to the candidate policies explored; however, it is not guaranteed to find a policy if one exists.

In (Shengyi Huang and Ontañón, 2022) an approach that enables action masking in deep RL is proposed. Action masking consists in disabling all the actions that would lead to a safety violation (Krasowski, Thumm et al., 2023). Shengyi Huang and Ontañón give theoretical foundations for the use of action masking in policy gradient methods, and show how it generates a valid policy gradient. Using a simplified real-time strategy game they show experimentally that invalid action masking scales better with respect to the size of the action space than *invalid action penalty*, which penalizes agents for taking invalid actions. Kanervisto et al. (2020) classify action masking (which they call “*remove actions*”) as a category of “*action space shaping*” approaches. They present ablation studies in game domains, showing the positive effect of action masking during learning on sample efficiency.

Finally, in (Varricchio, Alechina, Dastani, De Giacomo et al., 2024), we have proposed an approach that employs pure-past linear-time temporal logic (PPLTL) (De Giacomo, Di Stasio et al., 2020) to enforce safety constraints via action masking. Action selection is constrained based on the truth values of PPLTL formulas, which are computed at each timestep before the agent selects the action to perform. In Chapter 7 we will specifically delve into this work.

4

Maximally Permissive Reward Machines

In this chapter, we focus on the problem of synthesizing reward machines from a planning perspective. The central concept in planning is, trivially, that of *plans*. A plan can be a sequence of actions (*sequential plan*) or a multiset of actions with a partial order over them (*partial order plan*) that can be used to achieve some task in an environment. One might notice that this is very close to reinforcement learning’s goal; indeed, reinforcement learning can be thought of as a *heuristic* approach to planning. This stems from one, crucial, difference between planning and reinforcement learning: in planning, unlike in reinforcement learning, it is assumed that the dynamics of the environment are fully known and that the agent has access to them.

In (Illanes et al., 2019) reward machines were synthesised from a single plan. The approach assumed that an abstraction, in the form of a *planning domain* of the MDP in which the agent acts is provided. Here, by “*abstraction*”, we mean that the planning domain represents the MDP but abstracting away details. For example, consider the MDP of the OFFICEWORLD domain (Example 2.1). While in the MDP states encode the agent’s coordinates and actions directly modify these coordinates, we can define a planning domain in which the state encodes only in which room the agent is and whether it has coffee and/or mail, and actions only consist in moving to different rooms or collecting/delivering the coffee and mail. Starting from a planning domain, which specifies also a task, Illanes et al. propose to obtain a plan, either a sequential or partial order one, and then synthesise a reward machine from it. Expanding upon this idea, we proposed to synthesise reward machines from all the possible partial order plans that achieve the given task. In this chapter, we will show how this is possible, and provide experimental results comparing our approach to that of Illanes et al. As we will see, generating reward machines that encode multiple plans will lead to agents that learn better policies at the cost of longer training times.

The contents of this chapter are based on (Varricchione, Alechina, Dastani and Logan, 2024), which was published in the proceedings of the 27th European Conference on Artificial Intelligence (ECAI 2024).

4.1 Introduction

Reward machines were introduced in (Toro Icarte, Klassen, Valenzano and McIlraith, 2018) as a way of defining temporally extended (i.e., non-Markovian relative to the environment) tasks and behaviors. A *reward machine* (RM) is a Mealy machine where states represent abstract ‘steps’ or ‘phases’ in a task, and transitions correspond to observations of *high-level events* in the environment indicating that an abstract step/phase in the task has (or has not) been completed (Camacho, Toro Icarte et al., 2019; Toro Icarte, Klassen, Valenzano and McIlraith, 2022). The RM-based algorithm proposed in (Camacho, Toro Icarte et al., 2019) has been shown to out-perform state-of-the-art RL algorithms, especially in tasks involving temporally extended behaviours. However, while learning with a reward machine is guaranteed to converge to an optimal policy *with respect to the reward machine*, in general RMs provide no guarantees that the resulting policy is optimal *with respect to the task* encoded by the reward machine. For example, a reward machine may specify that event e should be observed before event ℓ , while in some environment states, it may be more efficient to achieve ℓ before e . In general, for an RM-based policy to be optimal with respect to a task, the reward machine for the task must encode all possible ways the task can be achieved.

Another problem with reward machines is how to generate them. While a declarative specification in terms of abstract steps or phases in a task is often easier to write than a conventional reward function, specifying a reward machine for a non-trivial task is challenging and prone to errors. Reward machines can be computed from task specifications expressed in a range of goal and property specification languages, including LTL and LTL_f , in a straightforward way (Camacho, Toro Icarte et al., 2019). However, reward machines generated from an abstract temporal formula may not expose significant task structure. Writing more “informative” specifications can be challenging, and, moreover, may inadvertently over-prescribe the order in which the steps are performed. One way to address this problem, is to generate a reward machine from a high-level abstract description of the learning environment, using techniques such as AI planning (Illanes et al., 2019; Illanes et al., 2020), or (in a multi-agent setting) ATL model checking (Varricchione, Alechina, Dastani and Logan, 2023). For example, Illanes et al. (Illanes et al., 2019) consider a high-level model, in the form of a planning domain, of the environment in which the agent acts. They show how planning techniques can be used to synthesise a plan for a task, which is then used to generate a reward machine for that task. The reward machine is used to train a meta-controller for a hierarchical RL agent. The controller chooses which option (corresponding to an abstract action in the planning domain) to execute next. Their results indicate that an agent trained using a plan-based reward machine outperforms (is more sample efficient than) a standard HRL agent. They also show that reward machines based on partial-order plans outperform reward machines generated from sequential plans, arguing that this is because partial-order plans allow more ways of completing a task.

While the results presented by Illanes et al. are encouraging, their approach does not allow maximum flexibility to the agent, and thus cannot ensure learning an optimal policy for the task. The reward machine they generate is based on a single partial-order plan. In many cases, a goal may be achieved by different plans and each

plan might be more appropriate in different circumstances, e.g., depending on the agent’s location, or the resources available.

In this chapter we propose a new approach to synthesising reward machines which is based on the *set of partial-order plans for a goal*. We present an algorithm which computes the set of partial-order plans for a planning task, and give a construction for synthesising a *maximally permissive* reward machine (MPRM) from a set of partial-order plans. We prove that the expected discounted future reward of optimal policies learned using an MPRM is greater than or equal to that obtained from optimal policies learned using a RM synthesised from any single partial-order plan. We introduce a notion of the *adequacy* of planning domain abstractions, which intuitively characterises when a planning domain captures all the relevant features of an MDP, and prove that the expected reward of an optimal policy learned using an MPRM synthesised from a *goal-adequate* planning domain is the same as that of an optimal policy for the underlying MDP. Finally, we evaluate MPRMs using three tasks in the CRAFTWORLD environment (Andreas et al., 2017) used in (Illanes et al., 2019; Illanes et al., 2020), and show that the agent obtains higher reward than with RMs based either on a single partial-order plan or on a single sequential plan.

4.1.1 Preliminaries

In this section we present the necessary preliminaries on planning to make this chapter self-contained. Aside from the background notions on planning, we also introduce an algorithm used to compute the set of all partial-order plans that can achieve a given planning task.

A *planning domain* $\mathcal{D} = \langle \mathcal{F}, \mathcal{A} \rangle$, is a pair where $\mathcal{F}$ is a set of *fluents*, and $\mathcal{A}$ is a set of *planning actions*. *Planning states* are subsets $S \subseteq \mathcal{F}$, where a fluent is in S if and only if it is true in S . Actions $\alpha \in \mathcal{A}$ are tuples $\alpha = \langle pre^+, pre^-, eff^+, eff^- \rangle$ such that each element of α is a subset of $\mathcal{F}$, $pre^+ \cap pre^- = \emptyset$ and $eff^+ \cap eff^- = \emptyset$. The “*pre*” sets are the sets of “*preconditions*”, whereas the “*eff*” are the sets of “*effects*”, or “*postconditions*”. pre^+ are the fluents that must be true to perform the action, whereas pre^- those that must be false. Analogously, eff^+ are the fluents that are made true by the action, whereas eff^- those that are made false. Thus, an action α can be executed from a planning state S if and only if $pre^+ \subseteq S$ and $pre^- \cap S = \emptyset$. Executing action α in state S results in the new state $S' = (S \setminus eff^-) \cup eff^+$. Given an MDP M and a planning domain $\mathcal{D}$, we assume that $A \cap \mathcal{A} = \emptyset$, i.e., the planning actions are not the same as the actions the agent can perform in the MDP. Intuitively, the planning actions can be seen as *high-level* or abstract actions which correspond to sequences of actions in the MDP. For example, in a Minecraft-like scenario where the agent can move up, down, left, and right on a grid, a planning action might be “*get wood*” corresponding to a sequence of movement actions ending in a cell containing wood.

As an example of a planning domain, consider the CRAFTWORLD environment (Andreas et al., 2017) in which an agent moves in a grid and has to gather resources which can then be used to produce items.¹ For example, the agent can build a bridge in order to reach the gold ore. We assume the agent can build two different types of

¹CRAFTWORLD is based on the popular video game Minecraft, and was used as a test domain in (Illanes et al., 2019; Illanes et al., 2020).

bridge: an iron bridge or a rope bridge. The iron bridge requires gathering wood and iron and then processing them in a factory. The rope bridge requires gathering grass and wood and processing them in a toolshed. The corresponding planning domain $\mathcal{D}$ can be formalised as:

$$\begin{aligned}\langle \mathcal{F} = \{ & \text{has-wood, has-grass,} \\ & \text{has-iron, has-bridge} \}, \\ \mathcal{A} = \{ & \text{get-wood, get-grass, get-iron,} \\ & \text{use-factory, use-toolshed} \} \rangle\end{aligned}$$

The *get-x* actions have no preconditions, and only one positive postcondition, i.e., that *has-x* is true. The *use-factory* action has the preconditions *has-wood* and *has-iron*, the positive postcondition *has-bridge*, and the negative postconditions, *has-wood* and *has-iron*, i.e., the *use-factory* action makes a bridge, “consuming” the resources collected by the agent in the process. The *use-toolshed* action has *has-wood* and *has-grass* as preconditions, the positive postcondition *has-bridge*, and the negative postconditions, *has-wood* and *has-grass*.

A *planning task* is a triple $\mathcal{T} = \langle \mathcal{D}, \mathcal{S}_I, \mathcal{G} \rangle$, where $\mathcal{D}$ is a planning domain, $\mathcal{S}_I$ is the initial planning state, and $\mathcal{G} = \langle \mathcal{G}^+, \mathcal{G}^- \rangle$ is a pair containing two subsets of $\mathcal{F}$ which are disjoint. Any planning state $\mathcal{S}$ such that $\mathcal{G}^+ \subseteq \mathcal{S}$ and $\mathcal{S} \cap \mathcal{G}^- = \emptyset$ is a *goal state*. For example, the planning task to build a bridge is given by the domain $\mathcal{D}$ we have previously defined, the initial state $\mathcal{S}_I = \emptyset$, and the goal $\mathcal{G} = \langle \{\text{has-bridge}\}, \emptyset \rangle$.

A *sequential plan* $\rho = [\alpha_0, \dots, \alpha_n]$ for a planning task $\mathcal{T}$ is a sequence of planning actions $\alpha_i \in \mathcal{A}$ such that: (i) it is possible to execute them sequentially starting from $\mathcal{S}_I$, and (ii) by doing so, the planning domain reaches a goal state. For example, the following sequential plan allows the agent to produce a rope bridge:

$$[\text{get-wood, get-grass, use-toolshed}]$$

A *partial-order plan* (POP) $\bar{\rho} = \langle \mathcal{A}', \prec \rangle$ is a pair where $\mathcal{A}'$ is a multiset of actions from $\mathcal{A}$ and $\prec$ is a partial order over $\mathcal{A}'$ (Chapman, 1987; McAllester and Rosenblitt, 1991). We write $\alpha \prec \alpha'$ to denote $(\alpha, \alpha') \in \prec$, meaning that action α must be performed before action α' . For example, the following partial-order plan allows the agent to produce an iron bridge:

$$\begin{aligned}\bar{\pi}_{\text{iron-bridge}} = \\ \langle \{ & \text{get-wood, get-iron, use-factory} \}, \\ \{ & \text{get-wood} \prec \text{use-factory,} \\ & \text{get-iron} \prec \text{use-factory} \} \rangle\end{aligned}$$

Sequential plans are a special case of partial-order plans where $\prec$ is a total order and the multiset of actions contains no duplicated action. In general, a partial-order plan corresponds to a *set* of sequential plans, i.e., the set of all sequential plans that can be obtained by extending the partial order $\prec$ to a total order (referred to as a “*linearisation*” of the partial-order plan). Compared to sequential plans, partial-order plans allow the agent greater flexibility in choosing the order in which actions are

executed. While a sequential plan constrains the agent to follow the total order of the plan, with a partial-order plan the agent can perform any action α , so long as all actions α' such that $\alpha' \prec \alpha$ have already been executed.

Typically, given a planning task, a partial-order planner, e.g., (Nguyen and Kambhampati, 2001), returns a single partial-order plan $\bar{p} = \langle \mathcal{A}', \prec \rangle$. However, in general, a planning task can be achieved using multiple partial-order plans, i.e., plans $\bar{p}'$ where the actions in $\mathcal{A}'$ are ordered differently, or which use different multisets of actions.

Definition 4.1 (Set of all partial-order plans). The set of all partial-order plans for a planning task $\langle \mathcal{D}, \mathcal{S}_I, \mathcal{G} \rangle$, $\bar{\Pi}$, is the set of plans $\langle \mathcal{A}', \prec \rangle$ where $\mathcal{A}' \subseteq \mathcal{A}$ and any linearisation $[\alpha_0, \dots, \alpha_n]$ of $\mathcal{A}'$ consistent with $\prec$ results in a goal state $\mathcal{S}$, i.e., $\mathcal{G}^+ \subseteq \mathcal{S}$ and $\mathcal{G}^- \cap \mathcal{S} = \emptyset$.

It is straightforward to give an algorithm that returns the set of all partial-order plans $\bar{\Pi}$ for a planning task, see Algorithm 3. We assume the following definitions. $steps(\bar{p})$ is the multiset of actions in the plan $\bar{p}$ and $ord(\bar{p})$ is the set of ordering constraints. In addition, the algorithm maintains a set $links(\bar{p})$ of causal links of the form (α', f, α) where α' and α are steps and f is a literal in the postcondition of α' and in the precondition of α . Causal links record the reason for adding step α' to the plan (in order to establish precondition of α), and are used to generate ordering constraints. A step α'' threatens a causal link (α', f, α) if α'' makes f false. To resolve the threat α'' should be placed either before α' in the order, or after α . An ordering is consistent if it is transitive and does not contain cycles, i.e., there is no α_i, α_j such that $(\alpha_i \prec \alpha_j), (\alpha_j \prec \alpha_i) \in ord$. Given a planning action α , $pre(\alpha)$ is the set containing the positive and negative literals of the fluents appearing, respectively, in the sets pre^+ and pre^- of α , and $eff(\alpha)$ is the set of positive and negative literals appearing, respectively, in eff^+ and eff^- . A precondition f of a step α is termed open if there is no causal link $(\alpha', f, \alpha) \in links(\bar{p})$ establishing f . A plan is complete if it has no open preconditions. Initially, the set of plans is empty, and $\bar{p}$ is initialised to a plan consisting of two steps: *start* and *finish*: *start* has no preconditions and the initial state $\mathcal{S}_I$ as a postcondition; *finish* has no postconditions and the goal $\mathcal{G}$ as a precondition. ord contains the single ordering constraint $\{start \prec finish\}$, and $links$ is empty.

The procedure POP-PLAN takes a partial-order plan $\bar{p}$ as input. If $\bar{p}$ has no open preconditions, i.e., the plan is complete, then we remove the steps *start* and *finish*, add it to the set of plans, and POP-PLAN returns (lines 5-6). Otherwise, we iterate over each open precondition in the set of open preconditions, *open* (lines 8-14). For each open precondition f , an action α' from the set of actions $\mathcal{A}$ of the planning domain is chosen which establishes f (line 9; if there are no actions which establish f , i.e., the plan cannot be extended to a complete plan, this branch of the computation terminates and $\bar{p}$ is discarded). The procedure ORDER is then called (line 14) to resolve any threats introduced by the addition of α' . If there are no threats, then POP-PLAN is called again with the updated $\bar{p}$ containing α' (lines 23-24). Instead, if there exists at least a threat, the set of sets of ordering constraints *cons* (line 18) contains all possible ways of safeguarding each threatened link $(\alpha_i, \neg f, \alpha_j)$. For each such set of ordering constraints c , if c is consistent with the current ordering constraints in $ord(\bar{p})$, they are added to $ord(\bar{p})$, and POP-PLAN is called to extend the plan for each possible ordering

Algorithm 3: Compute the set of all partial-order plans

Input: planning task $\mathcal{T} = \langle \mathcal{D}, \mathcal{S}_I, \mathcal{G} \rangle$

```

1   $start \leftarrow \langle \emptyset, \emptyset, \mathcal{S}_I, \mathcal{F} \setminus \mathcal{S}_I \rangle$ ; /* Initialise planning action  $start$  */
2   $finish \leftarrow \langle \mathcal{G}^+, \mathcal{G}^-, \emptyset, \emptyset \rangle$ ; /* Initialise planning action  $finish$  */
3   $\bar{\rho} \leftarrow \langle \{start, finish\}, \{start \prec finish\} \rangle$ ;
4   $\bar{\Pi} \leftarrow \emptyset$ ;
5  procedure POP-PLAN( $\bar{\rho}$ )
6     $open \leftarrow \text{open preconditions} \in \text{steps}(\bar{\rho})$ ;
7    if  $open = \emptyset$  then
8       $\bar{\Pi} \leftarrow \bar{\Pi} \cup \{ \langle \text{steps}(\bar{\rho}) \setminus \{start, finish\}, \text{ord}(\bar{\rho}) \setminus \{start \prec finish\} \rangle \}$ ;
9    else
10     for  $\alpha \in \text{steps}(\bar{\rho})$  s.t.  $f \in \text{pre}(\alpha) \wedge f \in open$  do
11       for  $\alpha' \in \mathcal{A} \cup \text{steps}(\bar{\rho})$  s.t.  $f \in \text{eff}(\alpha')$  do
12         if  $\alpha'$  is new then
13            $\bar{\rho} \leftarrow \langle \text{steps}(\bar{\rho}) \cup \{\alpha'\}, \text{ord}(\bar{\rho}) \cup \{start \prec \alpha' \prec finish\} \rangle$ ;
14         end
15          $\text{ord}(\bar{\rho}) \leftarrow \text{ord}(\bar{\rho}) \cup \{\alpha' \prec \alpha\}$ ;
16          $\text{links}(\bar{\rho}) \leftarrow \text{links}(\bar{\rho}) \cup \{(\alpha', f, \alpha)\}$ ;
17         ORDER( $\bar{\rho}, \alpha', f, \alpha$ );
18       end
19     end
20   end
21 end
22 procedure ORDER( $\bar{\rho}, \alpha', f, \alpha$ )
23    $\text{threats} \leftarrow \{(\alpha_i, \alpha_j) \mid (\alpha_i, \neg f, \alpha_j) \in \text{links}(\bar{\rho})\}$ ;
24   if  $\text{threats} \neq \emptyset$  then
25      $\text{cons} \leftarrow \{ \{o_1, \dots, o_n\} \mid (\alpha_j, \alpha_k)_i \in \text{threats} \wedge o_i = \alpha \prec \alpha_j \text{ or } o_i = \alpha_k \prec \alpha' \}$ ;
26     for  $c \in \text{cons}$  do
27       if  $\text{ord}(\bar{\rho}) \cup c$  is consistent then
28          $\text{ord}(\bar{\rho}) \leftarrow \text{ord}(\bar{\rho}) \cup c$ ;
29         POP-PLAN( $\bar{\rho}$ );
30       end
31     end
32   else
33     POP-PLAN( $\bar{\rho}$ );
34   end
35 end

```

of actions (lines 19-22). When a plan is found, we backtrack and continue from the ‘closest’ enclosing **for** loop (which may be iterating over sets of ordering constraints in ORDER, or actions α' and open preconditions f in POP-PLAN) to search for alternative ways of extending the incomplete plan $\bar{\rho}$. Algorithm 3 runs in EXPSPACE, as the set

of partial order plans is in the worst case exponential in the number of actions in $\mathcal{A}$. In practice, this is often not an issue: the planning domain is an abstraction of the underlying MDP, and the number of actions is typically small.

For the bridge task, the algorithm would produce another partial-order plan, in which the agent builds a rope bridge using grass:

$$\begin{aligned}\bar{\rho}_{\text{rope-bridge}} = \\ \langle \mathcal{A} = \{\text{get-wood}, \text{get-grass}, \text{use-toolshed}\}, \\ \prec = \{\text{get-wood} \prec \text{use-toolshed}, \\ \text{get-grass} \prec \text{use-toolshed}\} \rangle\end{aligned}$$

Thus giving us the set of all partial-order plans for the bridge task: $\bar{\Pi}_{\text{bridge}} = \{\bar{\rho}_{\text{iron-bridge}}, \bar{\rho}_{\text{rope-bridge}}\}$. In the supplementary material of this chapter (Section 4.5), we also provide all sequential plans that can be obtained by linearising the POPs in $\bar{\Pi}_{\text{bridge}}$.

4.2 Maximally Permissive Reward Machines

In this section, we show how the set of all partial-order plans, $\bar{\Pi}$, for a planning task $\mathcal{T} = \langle \mathcal{D}, \mathcal{S}_I, \mathcal{G} \rangle$, can be used to synthesise a reward machine $\mathcal{R}_{\bar{\Pi}}$ that is *maximally permissive*, i.e., which allows the agent maximum flexibility in learning a policy.

Let Π be the set of all linearisations ρ of all the partial-order plans in $\bar{\Pi}$. We denote by $\text{pref}(\rho)$ the set of all proper prefixes (of arbitrary length) of $\rho \in \Pi$. Note that the prefixes are finite, as the set of actions in $\bar{\Pi}$ is finite. Then, let $\text{states}(\text{pref}(\rho))$ be the set of sequences of planning states that is induced by the prefixes in $\text{pref}(\rho)$, assuming that the initial planning state is $\mathcal{S}_I$. We denote with $\text{steps}(\rho)$ the set of actions in a sequential plan, and with $\text{post}(\mathcal{A}')$ the set containing $\text{post}(a)$, as defined in Section 4.1.1, for each planning action $a \in \mathcal{A}'$. For an arbitrary sequence of planning states u , we denote with $\text{last}(u)$ the last element of the sequence. For sets of literals P of fluents in $\mathcal{F}$, we respectively denote with $P^+ \subseteq \mathcal{F}$ and $P^- \subseteq \mathcal{F}$ the sets of fluents with positive and negative literals in P .

Construction 4.1 (Maximally Permissive Reward Machine (MPRM)). *Fix the set of all partial-order plans $\bar{\Pi} = \{\bar{\rho}_1, \dots, \bar{\rho}_n\}$ for some planning task $\mathcal{T} = \langle \mathcal{D}, \mathcal{S}_I, \mathcal{G} \rangle$. Then, $\mathcal{R}_{\bar{\Pi}}$, the maximally permissive RM corresponding to $\bar{\Pi}$, is defined as follows:*

- $U = \left[\bigcup_{\rho \in \Pi} \text{states}(\text{pref}(\rho)) \right];$
- $u_0 = [\mathcal{S}_I];$
- $F = \{u_g\};$
- $\Sigma = \bigcup_{\rho \in \Pi} \text{post}(\text{steps}(\rho));$
- $\delta_u(u, P) = u\mathcal{S}$, where $\mathcal{S} = (\text{last}(u) \setminus P^-) \cup P^+$ and $u\mathcal{S} \in \text{states}(\text{pref}(\rho))$ for some linearisation $\rho \in \Pi$, or $= u_g$ if $\mathcal{G}^+ \subseteq \mathcal{S}$ and $\mathcal{G}^- \cap \mathcal{S} = \emptyset$;

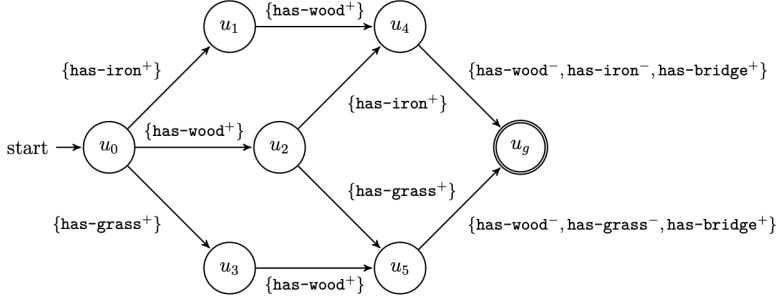


Figure 4.1: MPRM for the bridge task. Positive and negative postconditions are respectively denoted with a superscript $+$ and $-$.

$$\bullet \quad \delta_r(u, u') = \begin{cases} 0 & \text{if } u' = u_g \\ -1 & \text{otherwise.} \end{cases}$$

In the construction of the MPRM, the set of states correspond to the set of all possible prefixes of planning states across all POPs in the set used to build the reward machine. Then, the RM transitions from a state u to a state $u' = uS$ when it observes the set of literals P which are exactly the conditions such that $S = (\text{last}(u) \setminus P^-) \cup P^+$ and, most importantly, uS is a prefix of some linearisation of a POP in $\bar{\Pi}$. As soon as the RM “reaches” a sequence of states such that the last state is a goal state for the task (meaning also that a linearisation has been “completed”), it gives a reward of 0 to the agent and terminates in the unique final state u_g , while for all other transitions the agent gets a reward of -1 . Note that if uS is not the prefix of any linearisation of a POP in $\bar{\Pi}$, or S is not a goal state, then $\delta_u(u, P) = u$. Figure 4.1 shows the MPRM synthesised from the set $\bar{\Pi}_{\text{bridge}}$ of all partial-order plans for the bridge example we gave in Section 4.1.1.

In the remainder of this section, we provide a theoretical analysis linking the optimal policies that can be learned by an agent depending on the kind of reward machine it is equipped with. We consider RMs that can be built from the set of all partial-order plans (RM- $\bar{\Pi}$), a single partial-order plan (RM- $\bar{\rho}$), and a single sequential plan (RM- ρ), over the same planning domain $\mathcal{D}$. All RMs issue a non-negative reward only in the final state. We denote the optimal policy learnt using the set of all partial-order plans by $\pi_{\text{RM-}\bar{\Pi}}^*$, using a single partial-order plan by $\pi_{\text{RM-}\bar{\rho}}^*$, and using a single sequential plan by $\pi_{\text{RM-}\rho}^*$.

Theorem 4.1. *Let M be a labelled MDP, $\mathcal{D}$ a planning domain over M , and RM- $\bar{\Pi}$, RM- $\bar{\rho}$ and RM- ρ final state reward machines generated from $\mathcal{D}$ for the same task. Then,*

$$\pi_{\text{RM-}\bar{\Pi}}^* \geq \pi_{\text{RM-}\bar{\rho}}^* \geq \pi_{\text{RM-}\rho}^*$$

where recall that $\pi_1 \geq \pi_2$ if and only if $v(\pi_1(s)) \geq v(\pi_2(s))$ for all states $s \in \mathcal{S}$ of M .

Proof. The proof follows from the fact that any policy that can be learned using an RM synthesised from a single sequential plan can also be learned using an RM

synthesised from a partial-order plan which has the sequential plan as its linearisation (if it remains an optimal policy). Similarly, a policy learned using an RM synthesised from a single partial-order plan can also be learned using an RM that is synthesised from the set of all partial-order plans. $\square$

Theorem 4.1 shows that MPRMs allow an agent to learn an optimal policy with respect to the planning domain and the planning task. A natural question to ask is whether it learns a *goal-optimal* policy π^* , i.e., a policy that achieves the goal using the smallest number of actions in the underlying MDP. For example, an agent using Q-learning is guaranteed to learn a goal-optimal policy on an MDP where the agent is always given the same negative reward and the discount factor γ is exactly 1.

An agent using an MPRM will learn a goal-optimal policy if the planning domain and labelling are “adequate” for the goal.

Definition 4.2. Given a labelled MDP, planning domain $\mathcal{D}$, and goal $\mathcal{G}$, we say that $\mathcal{D}$ is *adequate* for $\mathcal{G}$ if, and only if:

- $\mathcal{G}$ corresponds to a set of planning domain fluents, i.e., $\mathcal{G} \subseteq \mathcal{F}$;
- a goal-optimal policy encounters all the state labels in some plan $\bar{\rho} \in \bar{\Pi}$ for $\mathcal{G}$, in the order consistent with the order in $\bar{\rho}$.

For example, if any policy to build a bridge has to encounter labels corresponding to getting wood, getting iron and using a factory, a planning domain and labelling containing only these fluents is adequate for the goal of having a bridge. However, if there is an alternative way of building a bridge that involves getting grass, and this label is missing in the planning domain, then the domain is not adequate for the goal of building a bridge.

Theorem 4.2. $\pi^* = \pi_{RM-\bar{\Pi}}^*$ if RM- $\bar{\Pi}$ is synthesized from a goal-adequate planning domain.

Proof. From the definition of a planning domain adequate for the goal, any goal-optimal policy has to go through the way-points encoded in the reward machine. $\square$

4.3 Empirical Evaluation

In this section, we evaluate maximally permissive reward machines in three tasks in the CRAFTWORLD environment, and show that the agent obtains higher reward with an MPRM than with RMs based on a single partial-order plan or a single sequential plan. In the first task, the agent has to build a bridge, as in the example in Section 4.1.1. For the second task, the agent has to collect gold. In the third task, the agent has to collect gold or a gem, and the task is considered achieved when the agent collects at least one of the two items. For the gold-or-gem task we have to slightly modify the definition of goal states in planning tasks: the goal is the pair $\mathcal{G} = \langle \mathcal{G}^+ = \{\text{has-gold}, \text{has-gem}\}, \mathcal{G}^- = \emptyset \rangle$, and a planning state $\mathcal{S}$ is a *goal* state if and only if $\mathcal{G}^+ \cap \mathcal{S} \neq \emptyset$ and $\mathcal{G}^- \cap \mathcal{S} = \emptyset$. The gold and the gem are collected as described in (Andreas et al., 2017): gold is collected by using a(ny) bridge, whereas the gem is collected using an axe. To produce an axe, the agent must combine a

stick, which can be obtained by processing wood at the workbench, with iron at the toolshed. We refer to these, respectively, as the “*bridge task*”, “*gold task*”, and “*gold-or-gem task*”. In the planning domain, we add the following fluents:

- For the gold task: `has-gold`;
- For the gold-or-gem task: `has-gold`, `has-stick`, `has-axe`, `has-gem`;

and the following planning actions:

- For the gold task:
 - `get-gold`, with one positive precondition, `has-bridge`, and one positive postcondition, `has-gold`;
- For the gold-or-gem task:
 - `use-workbench`, with one positive precondition, `has-wood`, one positive postcondition, `has-stick`, and one negative postcondition, `has-wood`;
 - `use-toolshed-for-axe`, with positive preconditions, `has-stick` and `has-iron`, one positive postcondition, `has-axe`, and two negative postcondition, `has-stick` and `has-iron`;
 - `get-gem`, with one positive precondition, `has-axe`, and one positive postcondition, `has-gem`;

The two partial-order plans for the bridge task $\{\bar{\rho}_{\text{iron-bridge}}, \bar{\rho}_{\text{rope-bridge}}\}$ that are in $\bar{\Pi}_{\text{bridge}}$ are given in Section 4.1.1. For the gold task, we extend $\bar{\rho}_{\text{iron-bridge}}$ and $\bar{\rho}_{\text{rope-bridge}}$ by adding the `get-gold` action, and by having `use-factory` $\prec$ `get-gold` and `use-toolshed` $\prec$ `get-gold`. For the gold-or-gem task, the set of partial-order plans consists of $\bar{\rho}_{\text{iron-bridge}}$, $\bar{\rho}_{\text{rope-bridge}}$ and $\bar{\rho}_{\text{gem}}$ in which the agent makes an axe and uses it to mine the gem. $\bar{\rho}_{\text{gem}}$ is defined as follows:

$$\begin{aligned} \bar{\rho}_{\text{gem}} = & \langle \mathcal{A} = \{\text{get-wood}, \text{get-iron}, \text{use-workbench}, \\ & \text{use-toolshed-for-axe}, \text{get-gem}\}, \\ \prec = & \{\text{get-wood} \prec \text{use-workbench}, \\ & \text{get-iron} \prec \text{use-toolshed-for-axe}, \\ & \text{use-workbench} \prec \\ & \text{use-toolshed-for-axe}, \\ & \text{use-toolshed-for-axe} \prec \text{get-gem}\} \rangle \end{aligned}$$

For each task, we also generate all sequential plans that can be obtained by linearising the POPs that can be used to achieve the task. Thus, for both the bridge and gold tasks, there are a total of 4 sequential plans and 2 partial-order plans. For the gold-or-gem task, there are a total of 7 sequential plans and 3 partial-order plans. In the Appendix, we provide formal definitions of the planning domains, and give also the plans for the gold and gold-or-gem tasks.

4.3.1 Experimental Setup

The maximally permissive RMs for each task were synthesised using the construction given in Section 4.2. The RMs for each partial-order and sequential plan were generated using the approach presented in (Illanes et al., 2019). Training is carried out by using Q-learning over the resulting MDP RMs (Toro Icarte, Klassen, Valenzano and McIlraith, 2022).²

For each task we generated 10 different maps of size 41 by 41 cells. The maps and initial locations were chosen so that from some locations a task can be completed more quickly by following a particular sequential plan. For example, in the first map for the bridge task, if the agent starts from a location in the upper half of the map (i.e., in the first 20 rows) it is more convenient to build an iron bridge, while in the lower half of the map it is more convenient to build a rope bridge. The MDP reward function r returns -1 for each step taken by the agent, until it achieves the task or the episode terminates. When the task is completed, the map and agent are “re-initialised”: the agent is placed on a random starting cell and its “inventory” is emptied, i.e., it contains no items. For each set of plans and single partial-order/sequential plan for a task, and for each of the 10 maps for the task, an agent was trained with the corresponding RM for 10,000,000 training steps. Training was carried out in episodes, lasting at most 1,000 steps, after which the environment was re-initialised regardless of whether the agent has achieved the task or not. Every 10,000 training steps the agent was evaluated on the same map used for training from 5 (predetermined) starting positions. We set the learning rate $\alpha = 0.95$, the discount rate $\gamma = 1$, and the exploration rate $\epsilon = 0.1$.

Our implementation, largely based off of that of (Illanes et al., 2019), is available on GitHub at github.com/giovannivarr/MPRM-ECAI24.

4.3.2 Results

For each approach, we plot the median and the 25th and the 75th percentiles (shaded areas) of the rewards obtained across all maps by the agents in the evaluations during training for each task. To make the plots more readable, we have “aggregated” results for the sequential and partial-order plan-based RMs: for each kind of plan we plot the median and the 25th and the 75th percentiles of all agents trained with an RM generated using that type of plan. In the plots, “QRM-MPRM” denotes the performance of the agent trained with a maximally permissive RM, while “Aggregated-QRM-Seq” and “Aggregated-QRM-POP” are, respectively, the aggregated performance of agents trained with sequential plan RMs and partial-order plan RMs. On the x -axis we plot the number of steps (in millions), while on the y -axis we plot the performance obtained during the evaluations run at the corresponding timestep.

Figure 4.2 shows the results for the bridge task, Figure 4.3 shows the results for the gold task, and Figure 4.4 shows the results for the gold-or-gem task. As can be seen, in all tasks the agent trained with the MPRM outperforms the aggregated

²Note that we do not provide results for a baseline that does not employ reward machines (e.g., Q-learning): as shown in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022), CRAFTWORLD is a complex environment with sparse rewards, making it infeasible for an agent to learn an effective policy without having access to a reward machine.

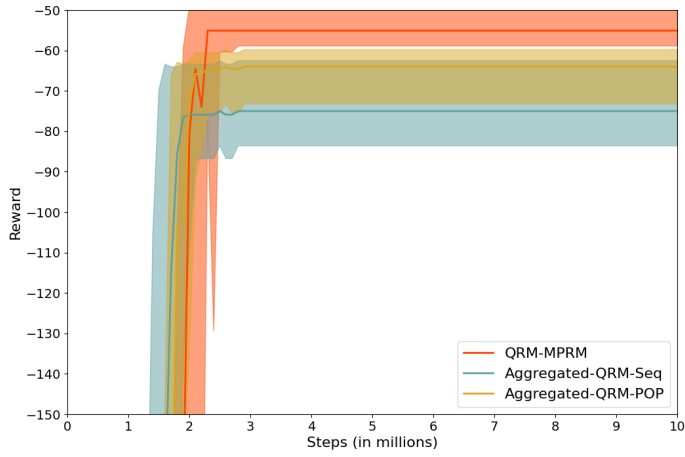


Figure 4.2: Results for the bridge task.

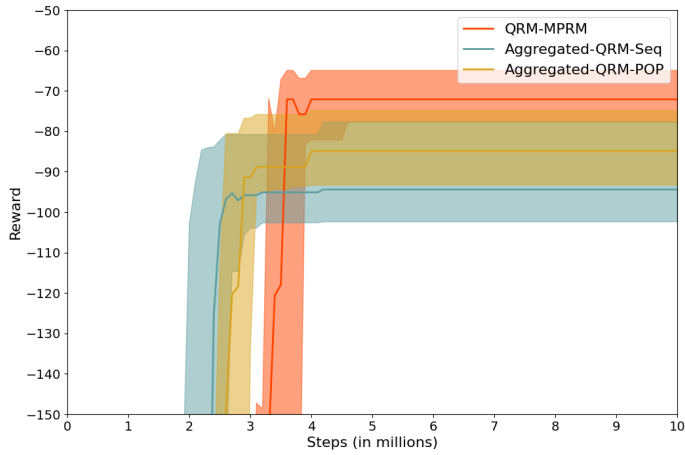


Figure 4.3: Results for the gold task.

results for the agents trained using RMs based on a single partial-order or sequential plan. This is as expected given Theorem 4.1. In addition, the agent trained using an RM based on a single partial-order plan outperforms the agent trained using an RM based on a single sequential plan. Again, this is consistent with Theorem 4.1 and

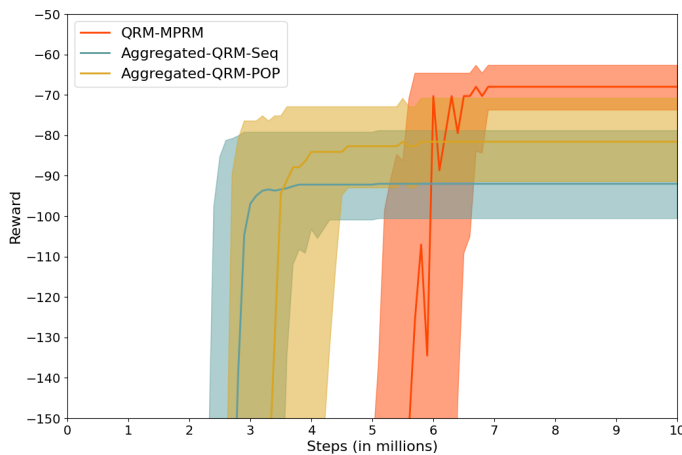


Figure 4.4: Results for the gold-or-gem task.

the results in (Illanes et al., 2019). However, in all experiments, the agent trained with the maximally permissive RM converges more slowly than the other agents (particularly in the gold-or-gem task). Intuitively, increasing the flexibility of the RM trades solution quality for sample complexity. Note that, as the planning domain used is *adequate* in the sense defined in Section 4.2 for all the tasks, the MPRM agent can learn an optimal policy for each task.

Figure 4.5 illustrates the behaviour of the agent trained using the MPRM and the agents trained using the RMs generated from each of the partial-order plans on a map for the gold-or-gem task. For readability, we have decided to not include agents trained with a sequential plan-based RM: note that none of them achieved the task in fewer steps than the agents shown in the figure. In the supplementary material we provide, for each agent, a file containing its trajectory (i.e., the ordered sequence of coordinates of the cells it visited) in the test, also for agents trained with a sequential plan-based RM. Given the initial position of the agent, the optimal plan for the task is to collect a gem (POP-2). As can be seen, both the agent trained using the POP-2-based RM and the MPRM agent achieve the goal by collecting a gem and complete the task in 60 steps. The agent trained using the RM based on the POP to collect gold using a rope bridge (POP-1) is also able to achieve the task, but in 63 steps. However, the agent trained using the RM based on the POP to collect gold using an iron bridge (POP-0) is unable to complete the task after 10,000,000 timesteps. This illustrates that the MPRM agent is able to choose the “correct” plan for the agent’s position, and the inherent problems of training agents using an RM based on a single plan.

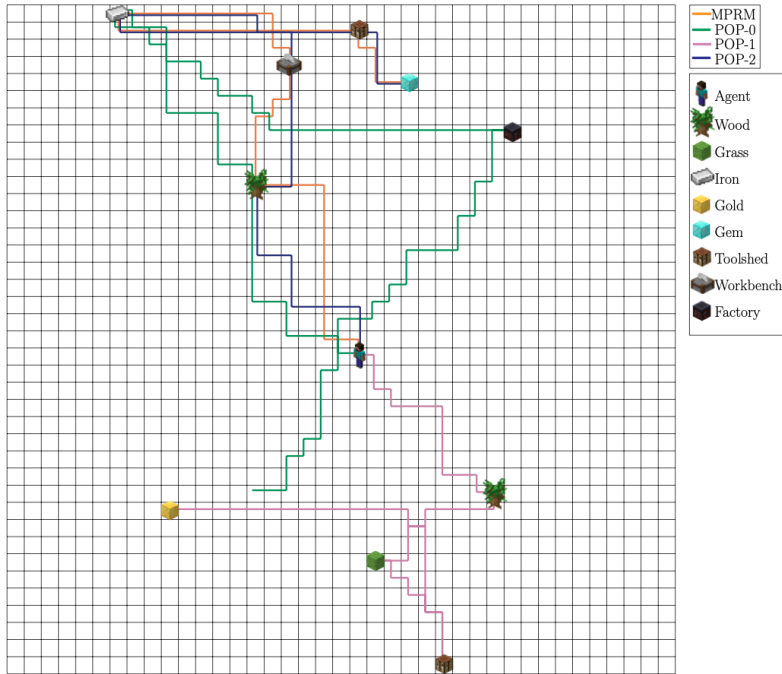


Figure 4.5: Illustration of the behaviour of the MPRM and POP-trained agents on the gold-or-gem task.

4.4 Conclusions

We have presented a new planning-based approach to synthesising maximally permissive reward machines which uses the set of partial-order plans for a goal rather than a single sequential or partial-order plan as in previous work. Planning-based approaches have the advantage that it is straightforward to train agents to achieve new tasks — given a planning domain, we can automatically generate a reward machine for a new task. We have provided theoretical results showing how agents trained using maximally permissive reward machines learn policies that are at least as good as those learned by agents trained with a reward machine built from an individual sequential or partial-order plan, and the expected reward of an optimal policy learned using an MPRM synthesised from a *goal-adequate* planning domain is the same as that of an optimal policy for the underlying MDP. Experimental results from three different tasks in the CRAFTWORLD environment suggest that these theoretical results apply in practice. However, our results also show that agents trained with maximally permissive RMs converge more slowly than agents trained using RMs based on a single plan. We believe this is because the increased flexibility of maximally permissive RMs trades solution quality for sample complexity. Our approach is therefore most useful when the quality of the resulting policy is paramount.

A limitation of our approach is that, in the worst case, the set of all partial order plans for a task may be exponential in the number of actions in the planning domain. In future work we would like to investigate the use of *top-k* planning techniques, e.g., (Katz and Lee, 2023), to sample a diverse subset of the set of all plans. Intuitively, such an approach could allow the quality of the resulting policy to be traded off against the number of plans in the sample.

Another line of future work is to investigate option-based approaches to learning (Sutton, Precup et al., 1999; Dietterich, 2000) as in (Illanes et al., 2020), where each abstract action in a plan is “implemented” as an option. We expect results similar to the ones in this chapter, where the agent trained with all partial-order plans is able to achieve a better policy but converging slower.

Finally, the experiments in Section 4.3 are limited to discrete environments. However, our approach is applicable to environments with continuous action and state spaces. Reward machines have previously been successfully applied in such environments (Toro Icarte, Klassen, Valenzano and McIlraith, 2022; Furelos-Blanco et al., 2023), and planning domains, which form the basis our approach, are agnostic about the underlying environment, as they are defined in terms of states resulting from (sequences of) MDP actions rather than the actions themselves. Nevertheless, learning in continuous environments is more challenging than learning in discrete ones, and evaluating the benefits of our approach in such environments is future work.

4.5 Supplementary Material

Sequential plans for the bridge task

Recall that the set of POPs for the bridge task is $\bar{\Pi}_{\text{bridge}} = \{\bar{\rho}_{\text{rope-bridge}}, \bar{\rho}_{\text{iron-bridge}}\}$. By linearising them, we obtain the following sequential plans:

$$\rho_0 = [\text{get-wood}, \text{get-iron}, \text{use-factory}]$$

$$\rho_1 = [\text{get-wood}, \text{get-grass}, \text{use-toolshed}]$$

$$\rho_2 = [\text{get-iron}, \text{get-wood}, \text{use-factory}]$$

$$\rho_3 = [\text{get-grass}, \text{get-wood}, \text{use-toolshed}]$$

Planning domain and plans for the gold task

The planning domain $\mathcal{D}_{\text{gold}} = \langle \mathcal{F}, \mathcal{A} \rangle$ is the following:

$$\mathcal{F} = \{\text{has-wood}, \text{has-grass}, \\ \text{has-iron}, \text{has-bridge}, \\ \text{has-gold}\}$$

$$\mathcal{A} = \{\text{get-wood}, \text{get-grass}, \text{get-iron}, \\ \text{use-factory}, \text{use-toolshed}, \\ \text{get-gold}\}$$

where each action has the following pre- and postconditions (recall that each tuple includes, in order, the set of positive and negative preconditions, and the set of positive and negative postconditions of the action):

$$\begin{aligned} \text{get-wood} &= \langle \emptyset, \emptyset, \{\text{has-wood}\}, \emptyset \rangle \\ \text{get-grass} &= \langle \emptyset, \emptyset, \{\text{has-grass}\}, \emptyset \rangle \\ \text{get-iron} &= \langle \emptyset, \emptyset, \{\text{has-iron}\}, \emptyset \rangle \\ \text{use-factory} &= \\ &\quad \langle \{\text{has-wood}, \text{has-iron}\}, \emptyset, \\ &\quad \{\text{has-bridge}\}, \{\text{has-wood}, \text{has-iron}\} \rangle \\ \text{use-toolshed} &= \\ &\quad \langle \{\text{has-wood}, \text{has-grass}\}, \emptyset, \\ &\quad \{\text{has-bridge}\}, \{\text{has-wood}, \text{has-grass}\} \rangle \\ \text{get-gold} &= \langle \{\text{has-bridge}\}, \emptyset, \{\text{has-gold}\}, \emptyset \rangle \end{aligned}$$

For the planning task we have $\mathcal{S}_I = \emptyset$, and $\mathcal{G} = \langle \{\text{has-gold}\}, \emptyset \rangle$. The set of all POPs $\bar{\Pi}_{\text{gold}}$ for this task is the following:

$$\begin{aligned} \bar{\rho}_{\text{iron-gold}} &= \\ &\quad \langle \mathcal{A} = \{\text{get-wood}, \text{get-iron}, \\ &\quad \text{use-factory}, \text{get-gold}\}, \\ &\quad \prec = \{\text{get-wood} \prec \text{use-factory}, \\ &\quad \text{get-iron} \prec \text{use-factory}, \\ &\quad \text{use-factory} \prec \text{get-gold}\} \rangle \\ \bar{\rho}_{\text{rope-gold}} &= \\ &\quad \langle \mathcal{A} = \{\text{get-wood}, \text{get-grass}, \\ &\quad \text{use-toolshed}, \text{get-gold}\}, \\ &\quad \prec = \{\text{get-wood} \prec \text{use-toolshed}, \\ &\quad \text{get-grass} \prec \text{use-toolshed}, \\ &\quad \text{use-toolshed} \prec \text{get-gold}\} \rangle \end{aligned}$$

By linearising the POPs in $\bar{\Pi}_{\text{gold}}$, we obtain the following sequential plans:

$$\begin{aligned} \rho_0 &= [\text{get-wood}, \text{get-iron}, \\ &\quad \text{use-factory}, \text{get-gold}] \\ \rho_1 &= [\text{get-wood}, \text{get-grass}, \\ &\quad \text{use-toolshed}, \text{get-gold}] \\ \rho_2 &= [\text{get-iron}, \text{get-wood}, \\ &\quad \text{use-factory}, \text{get-gold}] \end{aligned}$$

$$\rho_3 = [\text{get-grass}, \text{get-wood}, \\ \text{use-toolshed}, \text{get-gold}]$$

Planning domain and plans for the gold-or-gem task

The planning domain $\mathcal{D}_{\text{gold-or-gem}} = \langle \mathcal{F}, \mathcal{A} \rangle$ is the following:

$$\begin{aligned} \mathcal{F} = & \{\text{has-wood}, \text{has-grass}, \\ & \text{has-iron}, \text{has-bridge}, \\ & \text{has-stick}, \text{has-axe}, \\ & \text{has-gold}, \text{has-gem}\} \\ \mathcal{A} = & \{\text{get-wood}, \text{get-grass}, \text{get-iron}, \\ & \text{use-factory}, \text{use-toolshed}, \\ & \text{use-workbench}, \\ & \text{use-toolshed-for-axe}, \\ & \text{get-gold}, \text{get-gem}\} \end{aligned}$$

where each (new) action is defined as follows:

$$\begin{aligned} \text{use-workbench} = & \\ & \langle \{\text{has-wood}\}, \emptyset, \\ & \{\text{has-stick}\}, \{\text{has-wood}\} \rangle \\ \text{use-toolshed-for-axe} = & \\ & \langle \{\text{has-stick}, \text{has-iron}\}, \emptyset, \\ & \{\text{has-axe}\}, \{\text{has-stick}, \text{has-iron}\} \rangle \\ \text{get-gem} = & \langle \{\text{has-axe}\}, \emptyset, \{\text{has-gold}\}, \emptyset \rangle \end{aligned}$$

For the planning task, we have $\mathcal{S}_I = \emptyset$, and $\mathcal{G} = \langle \{\text{has-gold}, \text{has-gem}\}, \emptyset \rangle$. As described in Section 4, the task is considered achieved as soon as the agent collects one between the gold ore and the gem. Notice that there are no planning actions that allow the agent to collect the gold ore and the gem at the same time. Then, finding partial-order plans for this task is equivalent to taking the union of the partial-order plans for the planning tasks $\mathcal{T}_{\text{gold}} = \langle \mathcal{D}_{\text{gold-or-gem}}, \mathcal{S}_I, \langle \{\text{has-gold}\}, \emptyset \rangle \rangle$ and $\mathcal{T}_{\text{gem}} = \langle \mathcal{D}_{\text{gold-or-gem}}, \mathcal{S}_I, \langle \{\text{has-gem}\}, \emptyset \rangle \rangle$. Thus, we have that the set of partial-order plans is $\Pi_{\text{gold-or-gem}} = \{\bar{\rho}_{\text{rope-gold}}, \bar{\rho}_{\text{iron-gold}}, \bar{\rho}_{\text{gem}}\}$, where the first two POPs are as they were for the gold task, and the third POP is as it was presented in Section 4. The only POP that adds new sequential plans compared to the gold task is $\bar{\rho}_{\text{gem}}$; by linearising it we obtain the following sequential plans:

$$\rho_4 = [\text{get-wood}, \text{use-workbench}, \\ \text{get-iron}, \text{use-toolshed-for-axe}, \\ \text{get-gem}]$$

$\rho_5 = [\text{get-iron}, \text{get-wood},$
 $\text{use-workbench},$
 $\text{use-toolshed-for-axe}, \text{get-gem}]$

$\rho_6 = [\text{get-wood}, \text{get-iron},$
 $\text{use-workbench},$
 $\text{use-toolshed-for-axe}, \text{get-gem}]$

5 Synthesising Reward Machines for Cooperative Multi-Agent RL

We now move to *multi-agent* reinforcement learning. In multi-agent RL multiple agents learn to achieve their own tasks at the same time. Agents' tasks can be aligned towards a common goal, in which case the agents should cooperate in order to achieve their "*coalition*" task. In this case, we talk about "*cooperative* multi-agent reinforcement learning". In this chapter, we delve in this sub-field of RL and consider possible applications of reward machines in it.

While reward machines were originally defined for single-agent RL, they can also be used in multi-agent RL. (Neary et al., 2021) was the first work to ever use reward machines in a multi-agent setting. Their approach assumes that the coalition's task can be split in several subtasks, one per agent. Then, each agent is trained on its own, allowing for more efficient training. In this chapter, we show how to extend the approach of (Neary et al., 2021) by automatically synthesising the reward machine for the coalition of agents. This is done by using Alternating-time Temporal Logic, which can be used to determine whether a coalition of agents can bring about specific events and, if so, how the coalition can bring about such events. By automatically synthesising the reward machines we ensure that the reward machines used in the approach are devoid of human errors. Furthermore, we will also see how the obtained reward machines allow the agents to achieve performances which are comparable to that of agents trained with handmade reward machines.

The contents of this chapter are based on (Varricchione, Alechina, Dastani and Logan, 2023), which was published in the proceedings of the 20th European Conference on Multi-Agent Systems (EUMAS 2023), and on (Varricchione, Alechina, Dastani and Logan, 2026), an extension of the EUMAS paper which has been published at the Journal of Artificial Intelligence Research (JAIR).

5.1 Introduction

Reward machines (RMs) (Camacho, Toro Icarte et al., 2019; Toro Icarte, Waldie et al., 2019; Toro Icarte, Klassen, Valenzano and McIlraith, 2022) have recently been

proposed as a way of specifying rewards for reinforcement learning (RL) agents. RMs are Mealy machines used to specify tasks and rewards based on a high-level abstraction of the agent's environment. Providing an explicit encoding of the structure of the task has been shown to increase sample efficiency in reinforcement learning. For example, the RM-based algorithm proposed by Camacho, Toro Icarte et al. (2019) has been shown to out-perform state-of-the-art RL algorithms, especially in tasks requiring specific temporally extended behaviours.

Recently, Neary et al. (2021) proposed to use RMs as a means of specifying rewards for team tasks in multi-agent reinforcement learning. In cooperative multi-agent reinforcement learning (MARL) (Panait and Luke, 2005) the aim is to train a group of agents to perform a team task with the objective of maximising the expected future reward of the team. MARL is more challenging than single-agent RL. As the correctness of the actions of each agent may depend on the actions of other agents in the team, the agents must coordinate their actions (Boutilier, 1996). In addition, the agents are learning and updating their policies simultaneously. From the point of view of each individual agent, the learning problem is non-stationary; i.e., the optimal policy for each agent is constantly changing (Hernandez-Leal et al., 2019).

Neary et al. (2021) address these problems by specifying a multi-agent reward machine which encodes the abstract structure of the team task. The multi-agent reward machine is then decomposed into individual reward machines, one for each member of the team. The decomposition is carried out by projecting the coalition RM onto the set of observable events of each agent in the team. If the decomposition is done in such a way that the combined behaviour of the individual reward machines is “bisimulation equivalent” to that of the team reward machine, each agent can be trained using its individual reward machine to perform its part of the team task in a decentralised manner while still ensuring that the team task will be achieved by the joint actions of the agents. This avoids the problem of non-stationarity, and Neary et al. (2021) propose an algorithm based on this approach called “Decentralised Q-Learning with Projected Reward Machines” (DQPRM) which is shown to be more sample efficient than independent Q-learners (IQL) (Tan, 1993) and hierarchical independent learners (h-IL) (Tang et al., 2019).

However, Neary et al. (2021) assume that the multi-agent reward machine is given as input. An alternative approach to defining a reward machine by hand is to *synthesise* it from a high-level specification of the agent's task and environment. For example, Illanes et al. (2019) and Illanes et al. (2020) and Varricchione, Alechina, Dastani and Logan (2024) have used AI planning techniques to generate RMs for single agent learning tasks from a high-level goal the agent has to achieve and a high-level specification of the agent's environment specified in terms of fluents and STRIPS-like actions. In this approach, the high-level specification of the agent's task and environment are typically expressed in terms of high-level states, actions and fluents which abstract away the details of the agent's environment, i.e., high-level actions are typically “coarse-grained” and deterministic (“move from X to Y ”), while the actions available to the agent are, e.g., steering, accelerating, braking, etc., which may be nondeterministic.

In this chapter, we show how to synthesise reward machines for fully cooperative multi-agent reinforcement learning. Specifically, we synthesise the reward machines

from a high-level abstract description of the environment in which the agents act given as an epistemic concurrent game structure (ECGS), and a specification of the team task represented in a fragment of ATL, which we name “Flat Cooperative ATL” (FC-ATL), designed to express fully cooperative temporal multi-agent learning tasks. By using ATL model checking, we can generate an execution — a *witness* — of a coalition strategy that achieves the task in the high-level abstraction (if there is any). From the witness, we then synthesise the reward machine for the coalition of agents. The multi-agent reward machine is finally decomposed into multiple single-agent rewards machines, one for each agent in the coalition, which are then used to train individually the agents in the environment in which they act. As in the approach by Neary et al. (2021), the individual reward machines constitute a high-level abstract specification of the agent’s part of the team task. The corresponding individual strategies cannot be directly executed in the agents’ environment. However they guide the agents in learning which low-level actions should be performed to accomplish each high-level action in their part of the team task. We state constraints on the definition of abstract states and actions of the ECGS necessary for agents to be able to learn to realise a high-level action as sequences of MAE actions. Moreover, the use of Flat Cooperative ATL allows us to incorporate additional constraints on team goals, e.g., invariant properties, which are not considered by Neary et al. (2021).

Synthesising a reward machine from a high-level abstraction of the agent’s task and environment makes agents “taskable” (Illanes et al., 2020), as the user can define new agent tasks as goal conditions at the abstract level and a reward machine is automatically computed for the task. With the exception of C. Zhu et al. (2024), synthesis of RMs has been confined to the single-agent setting. The key idea underlying our approach is to extend RM synthesis to multi-agent settings and decompose the resulting multi-agent reward machine into individual reward machines allowing decentralised learning as done by Neary et al. (2021). To evaluate our approach, we present results from a number of benchmarks from (Neary et al., 2021) and Toro Icarte, Klassen, Valenzano and McIlraith (2022) which suggest that RMs synthesised with our automated approach perform as well or better in terms of sample efficiency than those defined by hand in those works. Moreover, we present results from a cooperative version of the continuous domain WATERWORLD introduced by Karpathy (2015), showing how our approach scales to more complex environments with continuous action- and state-spaces.

The remainder of the chapter is structured as follows. In Section 5.2 we introduce the necessary formal preliminaries that form the basis of our approach. In Section 5.3 we show how, given a multi-agent environment and an ATL formula specifying a team task, we can synthesise a reward machine for each agent in the team from a witness for the ATL formula. In Section 5.4 we present an evaluation of our automatically synthesised reward machines in four domains: COOPERATIVEBUTTONS and 10-Agent RENDEZVOUS from Neary et al. (2021), OFFICEWORLD from Toro Icarte, Klassen, Valenzano and McIlraith (2022), and WATERWORLD from Karpathy (2015). Finally, in Section 5.5 we conclude and outline directions for future work.

5.2 Preliminaries

In this section, we briefly introduce multi-agent reinforcement learning with reward machines and Alternating-time Temporal Logic, which form the basis of our approach.

5.2.1 Multi-Agent RL with RMs

We begin by defining a *multi-agent environment* (MAE) that specifies the low-level environment in which the agents act.

Definition 5.1 (Multi-Agent Environment). A multi-agent environment with n agents is a tuple $\langle \text{Agt}, (S_i)_{i \in \text{Agt}}, (A_i)_{i \in \text{Agt}}, T, (Prop_i)_{i \in \text{Agt}}, Val \rangle$ where:

- Agt is a non-empty finite set of n agents;
- S_i is the set of local states of agent i . We denote the set of joint states, i.e., the cartesian product of all the sets of states, with $\mathcal{S} = S_1 \times \dots \times S_n$;
- A_i is the set of actions of agent i . We denote the set of joint actions, i.e., the cartesian product of all the sets of actions, with $\mathcal{A} = A_1 \times \dots \times A_n$;
- $T : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the joint state transition probability distribution where $\Delta(\mathcal{S})$ is the set of all probability distributions over $\mathcal{S}$; we write $T(s'|s, a)$ for $T(s, a)(s')$, denoting the probability of transitioning from a joint state $s \in \mathcal{S}$ to a joint state $s' \in \mathcal{S}$ by performing a joint action $a \in \mathcal{A}$;
- $Prop_i$ is the finite set of propositional symbols “observable” by agent i , $Prop := \bigcup_{i \in \text{Agt}} Prop_i$ is the entire set of observable propositions. Note that the sets of propositional symbols observable by two agents i and j are not necessarily disjoint, i.e., it is possible that $Prop_i \cap Prop_j \neq \emptyset$ for $i, j \in \text{Agt}$;
- $Val : Prop \rightarrow 2^{\mathcal{S}}$ is a valuation function mapping each propositional symbol to the set of joint states in which it is true. For each agent i , we can also obtain its individual valuation function Val_i by taking the restriction of Val onto $Prop_i$ and S_i ;

A multi-agent environment specifies the dynamics of the low-level environment in which agents learn: the states and actions that are possible, and the probability of transitioning from a state s to a state s' on a (joint) action a . A MAE is similar to a Markov Game (Boutilier, 1996; Littman, 1994), i.e., a Markov Decision Process where the state and action spaces are factored to account for the fact that there are multiple agents acting in the environment. Unlike Markov Games, MAEs lack a reward function, as in our setting this will be specified by a reward machine.

A joint policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ maps any state in a MAE E to a probability distribution over the set of joint actions. We write $\pi(s, a)$ to denote the probability of performing the joint action a in the joint state s according to π .

Example 5.1. As an example of a MAE, consider the COOPERATIVEBUTTONS domain of Neary et al. (2021) shown in Figure 5.1. The COOPERATIVEBUTTONS environment

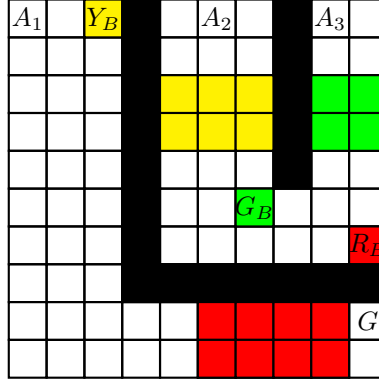


Figure 5.1: COOPERATIVEBUTTONS domain of Neary et al. (2021).

is a 10x10 grid world containing walls, three buttons, a goal location and three agents A_1 , A_2 and A_3 . Black walls are fixed, while the yellow, green and red walls can be lowered to allow an agent to pass by pressing the appropriate coloured buttons, Y_B , G_B and R_B . The task consists in allowing agent A_1 to reach the goal location G .

Agents can move left, up, right, down or stay in the same cell. Movement actions are nondeterministic: if an agent moves in any direction, it may end up in a cell adjacent to the one it was trying to reach with probability 0.2.

The set of propositional symbols is $Prop = \{Y_B, G_B, A_2^{R_B}, A_2^{-R_B}, A_3^{R_B}, A_3^{-R_B}, R_B, Goal\}$. Propositional symbols Y_B , G_B and R_B are true if, respectively, the yellow, green and red button has been pressed, while $Goal$ is true if an agent is on the goal location G . For the red button to be pressed, we require both agents A_2 and A_3 to press it together: if they are on the button, then propositional symbols $A_2^{R_B}$ and $A_3^{R_B}$ are true, if they leave the button then propositional symbols $A_2^{-R_B}$ and $A_3^{-R_B}$ are true. Agent A_1 's set of propositions is $Prop_{A_1} = \{Y_B, R_B, Goal\}$, agent A_2 's is $Prop_{A_2} = \{Y_B, G_B, A_2^{R_B}, A_2^{-R_B}, R_B\}$ and agent A_3 's is $Prop_{A_3} = \{G_B, A_3^{R_B}, A_3^{-R_B}, R_B\}$.

A joint state corresponds to the pair of coordinates $\langle x_j, y_j \rangle$ of each agent $A_j \in \{A_1, A_2, A_3\}$ and the set of propositions true in it.¹ Individual states for agent i contain only its coordinates and the set of propositions, from $Prop_i$, true in it.

Definition 5.2 (Single-Agent Environment). Given a MAE E , the single-agent environment (SAE) for agent $i \in Agt$ is the tuple $E_i = \langle S_i, A_i, T_i, Prop_i, Val_i \rangle$ where:

$$T_i(s, a, s') := \frac{\sum_{(s, a, s') \in S \times A \times S : s \in s, a \in a, s' \in s'} T(s, a, s')}{\sum_{(s, a, s') \in S \times A \times S : s \in s, a \in a} T(s, a, s')}$$

The function T_i is a probability measure over pairs $s \in S_i$, $a \in A_i$, i.e., $T_i(s, a, s') \geq 0$ for any $s' \in S_i$, and $\sum_{s' \in S_i} T_i(s, a, s') = 1$. Single-agent environments are used

¹For convenience, we will omit the set of propositional symbols true in joint states, and just give them as triples of coordinates. Whenever the set of propositional symbols is needed, we will explicitly state it beforehand.

to define the “part” of a MAE in which an agents learns its part of a team task independently of other agents.

In this chapter we specifically use labelling functions that output only sets of *literals*, thus specifying the truth values of the propositional symbols. Given a set of propositional symbols $Prop$, we denote with $\overline{Prop}$ the set of literals we derive from it. For a given propositional symbol $p \in Prop$, we denote with, respectively, p^+ and p^- its positive and negative literal.

Definition 5.3 (Labelling). Given a MAE² $E = \langle Agt, (S_i)_{i \in Agt}, (A_i)_{i \in Agt}, T, (Prop_i)_{i \in Agt}, Val \rangle$, its labelling is the, naturally induced, function $L : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow 2^{\overline{Prop}}$ mapping transitions in the MAE to the set of literals that hold after the transition. More precisely, given joint states $s, s' \in \mathcal{S}$ and joint action $a \in \mathcal{A}$, we have that $L(s, a, s') := \{p^+ \mid s \notin Val(p) \wedge s' \in Val(p)\} \cup \{p^- \mid s \in Val(p) \wedge s' \notin Val(p)\}$. As each agent $i \in Agt$ has its own set of observable propositional symbols $Prop_i$, we can define its individual labelling $L_i : S_i \times A_i \times S_i \rightarrow 2^{\overline{Prop_i}}$ by analogously taking $L_i(s_i, a_i, s'_i) := \{p^+ \mid s_i \notin Val_i(p) \wedge s'_i \in Val_i(p)\} \cup \{p^- \mid s_i \in Val_i(p) \wedge s'_i \notin Val_i(p)\}$.

In this chapter we focus on *postcondition labelling*, where $L(s, a, s')$ is the set of literals made true in s' by executing a in s . If, for a given atomic proposition $p \in Prop$, we have that neither of its literals appear in $L(s, a, s')$, then it means that its truth value has not changed from s to s' .

Example 5.1 (continued). Consider the previous example MAE COOPERATIVEBUTTONS with propositional symbols $Prop = \{Y_B, G_B, A_2^{R_B}, A_2^{-R_B}, A_3^{R_B}, A_3^{-R_B}, R_B, Goal\}$. As an example of a label for a transition, suppose the initial joint state s is $\langle \langle 1, 0 \rangle, \langle 5, 0 \rangle, \langle 8, 0 \rangle \rangle$ (with no propositional symbol being true) and that the joint action is $\langle \text{right}, \text{stay}, \text{stay} \rangle$. In this case, if agent A_1 correctly moves to the right, the next joint state will be $\langle \langle 2, 0 \rangle, \langle 5, 0 \rangle, \langle 8, 0 \rangle \rangle$, meaning that agent A_1 has correctly pressed the yellow button. Therefore, the transition $\langle \langle 1, 0 \rangle, \langle 5, 0 \rangle, \langle 8, 0 \rangle \rangle, \langle \text{right}, \text{stay}, \text{stay} \rangle, \langle \langle 2, 0 \rangle, \langle 5, 0 \rangle, \langle 8, 0 \rangle \rangle$ will be labelled with $\{Y_B^+\}$.

We can define tasks in multi-agent environment also via a reward machine. The labelling L connects the MAE E to the RM, by providing the RM with events (i.e., the set of literals) that occur as a result of transitions in the MAE. We therefore assume that the alphabet Σ of events corresponds to sets of literals built over the finite set of propositions $Prop$ of E , i.e., $\Sigma = 2^{\overline{Prop}}$.

Example 5.1 (continued). Consider again the COOPERATIVEBUTTONS domain with the labelling L introduced previously. Figure 5.2 shows the reward machine given by Neary et al. (2021) that makes use of the labelling L to describe the task of reaching the goal location G .

As agents have their own set of observable propositional symbols, we define the set of observable events of agent i as $\Sigma_i := 2^{\overline{Prop_i}}$. Similarly, for a coalition $A \subseteq Agt$, we define $\Sigma_A := 2^{\bigcup_{i \in A} \overline{Prop_i}}$. For a given event $e \in \Sigma$ and an agent i with its respective set of observable literals $\overline{Prop_i}$, we denote the restriction of e onto Σ_i by $e \upharpoonright \Sigma_i$, where

²The labelling of a SAE is obtained in an analogous manner.

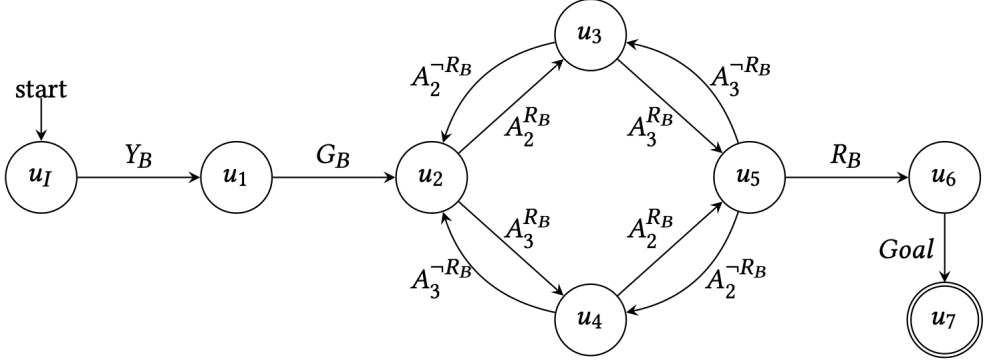


Figure 5.2: Reward machine by Neary et al. (2021) that describes the task of reaching the goal location in the COOPERATIVEBUTTONS domain.

$e \upharpoonright \Sigma_i = e \cap \overline{Prop_i}$. This will be used to define the “part of” the event e that is observable by a given subset of agents.

In what follows, we will refer to reward machines for coalitions as “coalition reward machines”, and to reward machines for individual agents as “individual reward machines”. Note that the two kinds of reward machines are identical from a formal point of view. However, intuitively, a coalition RM represents a coalition task, while each individual RM represents the contribution of a particular agent to the coalition task.

To formally define the *multi-agent reinforcement learning problem with reward machines*, we introduce the notion of a Markov game with a reward machine (MGRM). An MGRM is essentially a product of a multi-agent environment and a reward machine. It is similar to a Markov decision process with a reward machine (MDPRM) as defined by Toro Icarte, Klassen, Valenzano and McIlraith (2022), but generalised to the multi-agent setting, where the reward function is induced by a reward machine and the game is Markovian with respect to a set of states in the product.

Definition 5.4 (Markov Game with a Reward Machine). A Markov game with a reward machine with n agents is a tuple $G = \langle Agt, (S_i)_{i \in Agt}, (A_i)_{i \in Agt}, T, (Prop_i)_{i \in Agt}, Val, L, \gamma, U, u_0, F, \Sigma, \delta_u, \delta_r \rangle$ where:

- $Agt, S_i, A_i, T, (Prop_i)_{i \in Agt}, Val$ are as in Definition 5.1 (for all $1 \leq i \leq n$);
- $L : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow 2^{\overline{Prop}}$ is the labelling function of the underlying MAE, defined as in Definition 5.3;
- $\gamma \in [0, 1]$ is a discount factor;
- $U, u_0, F, \Sigma, \delta_u, \delta_r$ are as in Definition 2.3, with $\Sigma = 2^{\overline{Prop}}$; and
- if in states $s \in \mathcal{S}$, $u \in U$, the agents perform an action a and the MAE state changes from s to s' , then $u' = \delta_u(u, L(s, a, s'))$ and the agents receive reward $\delta_r(u, L(s, a, s'))$.

We define joint policies π as maps from pairs containing a MAE joint state s and a RM state u to probability distributions over the joint actions of the MAE, i.e., $\pi : S \times U \rightarrow \Delta(\mathcal{A})$. Analogously, an individual policy π is a map from pairs containing a SAE state s and a RM state u to probability distributions over the actions of the SAE, i.e., $\pi : S \times U \rightarrow \Delta(\mathcal{A})$.

Definition 5.5 (MGRM Learning Problem). For a given MGRM (MDPRM) G , the “MGRM learning problem” consists in learning an optimal joint policy $\pi^* : S \times U \rightarrow \Delta(\mathcal{A})$ that maximises the expected discounted future reward from any joint state of the MAE and RM state.

5.2.2 Alternating-Time Temporal Logic

Alternating-time Temporal Logic (ATL) (Alur et al., 2002) is a standard formalism for specifying the high-level behaviour of agents in multi-agent systems. The ATL formula plays a role similar to that of a planning goal in the synthesis of single-agent reward machines by Illanes et al. (2019) and Varricchione, Alechina, Dastani and Logan (2024). However, the use of ATL means we can easily specify more flexible properties; for example, that the agents can bring about ψ while maintaining ϕ , or can maintain some property forever, etc. Here we define the syntax and semantics of ATL with imperfect information. We need imperfect information because we do not assume that the agents can observe all the effects of each other’s actions, and it is important for decomposability that each agent bases its choice of actions only on what it can observe.

Let $Agt = \{i_1, \dots, i_n\}$ be a set of n agents and $Prop$ denote a set of propositions. Formulas of ATL are defined by the following syntax

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid \langle\langle A \rangle\rangle X\varphi \mid \langle\langle A \rangle\rangle G\varphi \mid \langle\langle A \rangle\rangle \varphi U \psi$$

where $p \in Prop$ is a proposition and $A \subseteq Agt$. Here, $\langle\langle A \rangle\rangle X\varphi$ means that a coalition A can ensure that the next state satisfies φ , $\langle\langle A \rangle\rangle G\varphi$ means that A has a strategy to make sure that φ is always true, and $\langle\langle A \rangle\rangle \varphi U \psi$ means that A has a strategy to eventually enforce ψ while maintaining the truth of φ before doing so.

The models of ATL are concurrent game structures. Imperfect information is modelled by indistinguishability relations between states, one for each agent. The resulting concurrent game structures are called “epistemic concurrent game structures” (ECGS).

Definition 5.6 (Epistemic Concurrent Game Structure). An epistemic concurrent game structure is a tuple $M = \langle Agt, Q, (Prop_i)_{i \in Agt}, V, (\sim_i)_{i \in Agt}, Act, d, \delta \rangle$ where:

- Agt is a non-empty finite set of n agents;
- Q is a non-empty finite set of states;
- $Prop_i$ is the finite set of propositional symbols “observable” by agent i , $Prop := \bigcup_{i \in Agt} Prop_i$ is the entire set of observable propositions;

- $V : Prop \rightarrow 2^Q$ is a valuation which associates each proposition in $Prop$ with a subset of states where it is true. We assume that two distinct states cannot have the same valuation, i.e., they cannot make the same subset of propositional symbols true;
- $\sim_i \subseteq Q \times Q$ for each $i \in Agt$ is an equivalence relation. For each state $q \in Q$, we denote with $[q]_i$ the equivalence class of q for $\sim_i$;
- Act is a non-empty finite set of actions;
- $d : Q \times Agt \rightarrow 2^{Act} \setminus \{\emptyset\}$ is a deterministic function which assigns to each $q \in Q$ a non-empty set of actions available to each agent $i \in Agt$, with the constraint that $q \sim_i q'$ implies that $d(q, i) = d(q', i)$. We denote joint actions by all agents in Agt available at q by $D(q) = d(q, i_1) \times \dots \times d(q, i_n)$;
- $\delta : (q, \sigma) \mapsto Q$ is a function that gives for every $q \in Q$ and joint action $\sigma \in D(q)$ the state resulting from executing σ in q . We write $q \xrightarrow{\sigma} q'$ to abbreviate $\delta(q, \sigma) = q'$.

Example 5.1 (continued). Let us now formalize the COOPERATIVEBUTTONS as an ECGS. The set of agents is $Agt = \{A_1, A_2, A_3\}$. The set of states is $Q = \{q_0, q_{\{Y_B\}}, q_{\{Y_B, G_B\}}, q_{\{Y_B, G_B, R_B\}}, q_{\{Y_B, G_B, R_B, Goal\}}\}$. For the observable propositions, we have that $Prop_{A_1} = \{Y_B, R_B, Goal\}$, $Prop_{A_2} = \{Y_B, G_B, R_B\}$, and $Prop_{A_3} = \{G_B, R_B\}$. A state q_X is in $V(p)$ for some proposition $p \in Prop$ if and only if $p \in X$. For each agent, $\sim_i$ is the relationship where, for any pair of states $q, q' \in Q$, $q \sim_i q'$ if, for any $p \in Prop_i$, $q \in V(p)$ if and only if $q' \in V(p)$. The set of actions is $Act = \{\text{press_yellow}, \text{press_green}, \text{press_red}, \text{to_goal}, \text{nil}\}$. $\text{nil} \in d(q, i)$ for any state $q \in Q$ and agent $i \in Agt$. For the other actions: $\text{press_yellow} \in d(q_0, A_1)$, $\text{to_goal} \in d(q_{\{Y_B, G_B, R_B\}}, A_1)$, $\text{press_green} \in d(q_{\{Y_B\}}, A_2)$, $\text{press_red} \in d(q_{\{Y_B, G_B\}}, A_2)$, and $\text{press_red} \in d(q_{\{Y_B, G_B\}}, A_3)$. Finally, for the transition function: $\delta(q_0, \sigma) = q_{\{Y_B\}}$ if the action of agent A_1 in σ is press_yellow , $\delta(q_{\{Y_B\}}, \sigma) = q_{\{Y_B, G_B\}}$ if the action of agent A_2 in σ is press_green , $\delta(q_{\{Y_B, G_B\}}, \sigma) = q_{\{Y_B, G_B, R_B\}}$ if the actions of agents A_2 and A_3 in σ are press_red , and $\delta(q_{\{Y_B, G_B, R_B\}}, \sigma) = q_{\{Y_B, G_B, R_B, Goal\}}$ if the action of agent A_1 in σ is to_goal .

Given an ECGS M , we denote the set of all infinite sequences of states (computations) by Q^ω . For a computation $\lambda = q_0 q_1 \dots \in Q^\omega$, we use the notation $\lambda[j] = q_j$. We denote joint actions with σ .

Given an ECGS M and a state $q \in Q$, a *joint action by a coalition* $A \subseteq Agt$ is a tuple $\sigma_A = (\sigma_i)_{i \in A}$ such that $\sigma_i \in d(q, i)$. The set of all joint actions for A at state q is denoted by $D_A(q)$. Given a joint action by the grand coalition $\sigma \in D(q)$, σ_A denotes the joint action executed by A : $\sigma_A = (\sigma_i)_{i \in A}$. The set of all possible outcomes of a joint action $\sigma_A \in D_A(q)$ at state q is denoted by $out(q, \sigma_A) = \{q' \in Q \mid \exists \sigma' \in D(q) : \sigma_A = \sigma'_A \wedge q \xrightarrow{\sigma'} q'\}$. Given an ECGS M , a *strategy for a coalition* $A \subseteq Agt$ is a mapping $\Pi_A : Q \rightarrow Act^{|A|}$ such that, for every $q \in Q$, $\Pi_A(q) \in D_A(q)$. A computation $\lambda \in Q^\omega$ is consistent with a strategy Π_A if and only if, for all $j \geq 0$, $\lambda[j+1] \in out(\lambda[j], \Pi_A(\lambda[j]))$. We denote by $out(q, \Pi_A)$ the set of all consistent computations λ of Π_A that start from

q . Given a coalition strategy Π_A for some coalition A , for any agent $i \in A$, we denote by Π_i the restriction of strategy Π_A to agent i .

As we consider postcondition labelling, we also “label” actions $\sigma_i \in Act$ (for any agent $i \in Agt$) with two subsets of propositional symbols $Prop_{\sigma_i}^+$ and $Prop_{\sigma_i}^-$ which are, respectively, the sets of propositional symbols true and false after σ_i is executed. Obviously, for any action σ_i , we must have that $Prop_{\sigma_i}^+ \cap Prop_{\sigma_i}^- = \emptyset$. For a joint action $\sigma = (\sigma_i)_{i \in Agt}$, $Prop_{\sigma}^+ = \bigcup_{i \in Agt} Prop_{\sigma_i}^+$ and $Prop_{\sigma}^- = \bigcup_{i \in Agt} Prop_{\sigma_i}^-$. We assume that the ECGS contains only joint actions such that for any distinct pair of agents $i, i' \in Agt$, $Prop_{\sigma_i}^+ \cap Prop_{\sigma_{i'}}^- = \emptyset$. Hence, transitions are defined so that $q \xrightarrow{\sigma} q'$ with: (i) $q' \in V(p)$ for any $p \in Prop_{\sigma}^+$, (ii) $q' \notin V(p)$ for any $p \in Prop_{\sigma}^-$, and (iii) for all $p \in Prop$ such that $p \notin Prop_{\sigma}^+ \cup Prop_{\sigma}^-$ we have that $q' \in V(p)$ if and only if $q \in V(p)$.

For a given agent ECGS action σ_i , $\overline{post(\sigma_i)} := \{p \mid p \in Prop^+(\sigma_i)\} \cup \{\neg p \mid p \in Prop^-(\sigma_i)\}$. Analogously, we define $\overline{post(\sigma)}$ for any joint action σ .

Some strategies are unrealistic in that they require agents to select different actions in two states that they cannot distinguish. For this reason, we will only consider so-called “uniform” strategies:

Definition 5.7 (Uniform strategy). A strategy for agent $i \in Agt$, Π_i , is *uniform* if and only if it specifies the same choices for indistinguishable situations: if $q \sim_i q'$ then $\Pi_i(q) = \Pi_i(q')$. A strategy for a coalition A is uniform if and only if it is uniform for each $i \in A$.

Example 5.1 (continued). We now give a uniform strategy Π for the coalition of agents in the COOPERATIVEBUTTONS task:

- $\Pi_{A_1}(q_0) = \text{press_yellow}$, $\Pi_{A_1}(q_{\{Y_B, G_B, R_B\}}) = \text{to_goal}$, and $\Pi_{A_1}(q) = \text{nil}$ for any other $q \in Q$;
- $\Pi_{A_2}(q_{\{Y_B\}}) = \text{press_green}$, $\Pi_{A_2}(q_{\{Y_B, G_B\}}) = \text{press_red}$, and $\Pi_{A_2}(q) = \text{nil}$ for any other $q \in Q$;
- $\Pi_{A_3}(q_{\{Y_B, G_B\}}) = \text{press_red}$, and $\Pi_{A_3}(q) = \text{nil}$ for any other $q \in Q$.

As can be checked, the strategy Π is uniform, as it satisfies the condition outlined in Definition 5.7 for each agent.

Strong uniformity requires, in addition, that in order for a formula of the form $\langle\langle A \rangle\rangle \varphi$ to be true in a state q , the same uniform strategy by A should ensure φ from all the states indistinguishable from q by A , i.e., for all states $q' \in [q]_A$, where $[q]_A$ is the equivalence class of q for $\sim_A := \bigcap_{i \in A} \sim_i$, we have that the uniform strategy that enforces φ from q also does from all $q' \in [q]_A$.

Given an ECGS M , a state q of M , the truth of an ATL formula φ with respect to M and q is defined inductively on the structure of φ as follows:

- $M, q \models p$ if and only if $q \in V(p)$;
- $M, q \models \neg \phi$ if and only if $M, q \not\models \phi$;
- $M, q \models \phi \vee \psi$ if and only if $M, q \models \phi$ or $M, q \models \psi$;

- $M, q \models \langle\langle A \rangle\rangle X \psi$ if and only if there exists a uniform strategy Π_A such that for all $q' \sim_A q$, for all $\lambda \in \text{out}(q', \Pi_A)$: $M, \lambda[1] \models \psi$;
- $M, q \models \langle\langle A \rangle\rangle G \phi$ if and only if there exists a uniform strategy Π_A such that for all $q' \sim_A q$, for all $\lambda \in \text{out}(q', \Pi_A)$ and $j \geq 0$: $M, \lambda[j] \models \phi$;
- $M, q \models \langle\langle A \rangle\rangle \phi U \psi$ if and only if there exists a uniform strategy Π_A such that for all $q' \sim_A q$, for all $\lambda \in \text{out}(q', \Pi_A)$, $\exists j \geq 0$: $M, \lambda[j] \models \psi$ and $M, \lambda[k] \models \phi$ for all $k \in \{0, \dots, j-1\}$.

We specify tasks for coalitions in a fragment of ATL, which we call “Flat Co-operative ATL” and abbreviate with “FC-ATL”. As we focus on fully cooperative multi-agent reinforcement learning, where the set of agents Agt cooperate to reach a common goal (and actions by the grand coalition are deterministic), FC-ATL contains only formulas of the form $\langle\langle A \rangle\rangle \varphi$ such that: (i) the coalition of agents A is always the grand coalition Agt , and (ii) φ contains no other coalitional modalities (and hence, temporal modalities). As $\langle\langle A \rangle\rangle \varphi$ represents the (single) team task the agents Agt will be trained to achieve, it suffices to consider the ATL fragment with only one coalitional modality and no disjunction or conjunction of coalitional formulas. (A conjunction or disjunction of coalitional formulas represents two different learning tasks.) Note that, even with a single coalitional modality we can still encode reachability and safety (including invariants) tasks.

The syntax of FC-ATL is given by:

$$\begin{aligned}\varphi_0 &::= p \mid \top \mid \neg \varphi_0 \mid \varphi_0 \vee \varphi_0 \\ \varphi_1 &::= \langle\langle \text{Agt} \rangle\rangle X \varphi_0 \mid \langle\langle \text{Agt} \rangle\rangle G \varphi_0 \mid \langle\langle \text{Agt} \rangle\rangle \varphi_0 U \varphi_0\end{aligned}$$

where only formulas generated by φ_1 are used to encode tasks for the grand coalition of agents.

Given the syntax of FC-ATL, in the remainder of the paper we use A to refer to the grand coalition Agt .

For example, the task from the COOPERATIVEBUTTONS domain in Example 5.1 can be specified as:

$$\langle\langle \text{Agt} \rangle\rangle \top U \text{Goal}$$

This formula is true if the grand coalition Agt has a uniform strategy to reach a state that satisfies *Goal*.

We note that we could have used a different formalism to specify tasks; for example, a fragment of Linear-time Temporal Logic (LTL) or of Computation Tree Logic (CTL) (Clarke and Emerson, 1981). We chose ATL over LTL and CTL because its ECGS-based semantics allow to express in a natural way the individual abilities of agents and what they can observe. Moreover, by model checking an FC-ATL formula, we can obtain the specific sequences of high-level ECGS actions that each agent should perform in order to achieve the given task. However, approaches based on synthesis for (fragments of) CTL and LTL could also have been used.³

We now define the *witness* of a (uniform) strategy for an ATL coalitional modality formula, given an ECGS and an initial state in it. Witnesses are used to define both

³We discuss the availability of tools which can perform ATL/LTL/CTL synthesis in Section 5.4.1.



Figure 5.3: Witness for the COOPERATIVEBUTTONS task.

coalition reward machines and individual reward machines, as we will see in the following section.

Definition 5.8 (Witness). Given an ECGS $M = \langle \text{Agt}, Q, (\text{Prop}_i)_{i \in \text{Agt}}, V, (\sim_i)_{i \in \text{Agt}}, \text{Act}, d, \delta \rangle$, a state $q \in Q$ and a coalition formula $\langle\langle A \rangle\rangle \varphi$ such that $M, q \models \langle\langle A \rangle\rangle \varphi$ using a uniform strategy Π_A for A , a *witness* $W(q, \Pi_A)$ for Π_A to achieve φ starting from q is defined as follows, depending on the form of φ :

- $\langle\langle A \rangle\rangle \varphi = \langle\langle A \rangle\rangle X \psi$: the witness is a tree of height 1 where the root is q , all outgoing edges are labelled with the action $\Pi_A(q)$, and the leaves are states $q' \in \text{out}(q, \Pi_A(q))$. Note that, in the case where $A = \text{Agt}$, there is a single leaf node $q \xrightarrow{\Pi_A(q)} q'$.
- $\langle\langle A \rangle\rangle \varphi = \langle\langle A \rangle\rangle \phi U \psi$: the witness is a tree where the root is q and each branch corresponds to the set of prefixes of λ in $\text{out}(q, \Pi_A)$, each prefix ends in the first state where ψ is true, and edges out of each q' are labelled by $\Pi_A(q')$. In the case where $A = \text{Agt}$, the witness tree has a single branch $q, \Pi_A(q), q_1, \Pi_A(q_1), \dots, \Pi_A(q_n), q_{n+1}$ of length $n + 1$ corresponding to the prefix of the single element λ of $\text{out}(q, \Pi_A)$, where $\lambda[n] = q_{n+1}$ is the first state on λ that satisfies ψ .
- $\langle\langle A \rangle\rangle \varphi = \langle\langle A \rangle\rangle G \phi$: the witness is a graph where the initial vertex is q and containing each computation $\lambda \in \text{out}(q, \Pi_A)$, ending in the first occurrence of a repeated state q_n . Edges out of each state q' are labelled by $\Pi_A(q')$. In the graph, for such repeated states q_n , there is a transition labelled by $\Pi_A(q_n)$ to states in $\text{out}(q_n, \Pi_A(q_n))$ that occur earlier in the graph (since q_n itself occurs earlier in the graph, its successors by $\Pi_A(q_n)$ are there as well). In the case where $A = \text{Agt}$, the witness graph contains the ECGS states $q, q_1, \dots, q_n$ from the unique computation $q, q_1, \dots, q_n, (q'_1, \dots, q'_m, q_n)^*$ (where $q_n, q'_1, \dots, q'_m, q_n$ is the first cycle encountered on the computation).

In Figure 5.3 we show the witness obtained from the uniform strategy for the COOPERATIVEBUTTONS task.

For formulas of the form $\langle\langle A \rangle\rangle \phi U \psi$ or $\langle\langle A \rangle\rangle X \psi$, we will refer to the states in the witness in which ψ becomes true as the “final states”, and use the notation q_f to refer to them. Formally, given an initial ECGS state q and a strategy Π for a formula of the form $\langle\langle A \rangle\rangle \phi U \psi$ or $\langle\langle A \rangle\rangle X \psi$, the set of final states of the witness is $Q_f(W(q, \Pi)) := \{q \in W(q, \Pi) \mid M, q \models \psi\}$.

Finally, we note that a witness is just a representation of a (uniform) strategy as a graph: it therefore does not affect how the strategy itself is defined or obtained.

5.2.3 Safe Abstractions and Learnable Tasks

We conclude this section by defining “safe abstractions” and “learnable tasks” with respect to a given MAE (SAE). Intuitively, safe abstractions are abstractions that

correctly represent how, via high-level events, an MAE (SAE) evolves; whereas learnable tasks are tasks expressed in a safe abstraction for which there exists a policy that achieves the task in the MAE (SAE).

In general, there are many possible abstractions of a given multi-agent environment E and its corresponding SAEs. For agents to be able to learn how to perform a task using a reward machine synthesised using an ECGS M , M must satisfy a number of properties. Fix a multi-agent environment E . We say that an ECGS state $q \in Q$ “abstracts” an MAE joint state $s \in S$ if and only if the sets of propositional symbols $\{p \mid q \in V(p)\}$ and $\{p \mid s \in Val(p)\}$ are equal, i.e., q and s satisfy the same propositional symbols.⁴ Note that this does not imply a bijection between the two: in distinct joint states of an MAE the same set of propositional symbols can be true, which is not the case in an ECGS.

Definition 5.9 (Safe abstraction). An ECGS M is a “safe abstraction” of a MAE E if, for any transition $q \xrightarrow{\sigma} q'$ in M , there is a joint policy $\pi_{q \xrightarrow{\sigma} q'}$ that “implements” the transition in E , i.e., if agents follow joint policy π from any joint state s in E that is abstracted by q , then E will eventually transition to some joint state s' that is abstracted by q' .

The notion of a safe abstraction is similar to standard techniques for producing existential abstractions of multi-agent systems, such as the one given by Cohen et al. (2009). It is also related to the notion of safe state abstraction in reinforcement learning introduced, for example, by Kokel et al. (2021); however, we do not consider rewards in our safe abstractions. In practice, safe abstractions ensure that for any high-level transition there is a low-level policy that allows the agents to “perform” the high-level transition in the low-level MAE. This means that the policy is able to enforce the same “changes” in the propositional symbols (i.e., the same “events”) that are caused by the high-level transition. This is crucial, as otherwise it might be the case that a high-level strategy for some task (computed from the ECGS) might not be executable in the low-level MAE if it uses a high-level transition which has no policy that implements it in the low-level MAE. Given this, we can define when a task is “learnable” by a coalition of agents A .

Definition 5.10 (Learnable tasks). Fix a MAE E and an ECGS M that is a safe abstraction of E . A task, expressed by an ATL formula $\langle\langle A \rangle\rangle\varphi$, is “learnable” by coalition A if and only if there is some ECGS state $q \in Q$ from which there is a uniform strategy for A , i.e., $M, q \models \langle\langle A \rangle\rangle\varphi$.

In the rest of the chapter, we will implicitly assume that the ECGS we are given is a safe abstraction of the input MAE, and that the task expressed by the input ATL formula is learnable with respect to such ECGS. This assumption ensures that, if there exists a strategy to achieve the task in the input ECGS, then there also exists a joint policy to achieve it in the MAE. The existence of a useful (small enough to admit model-checking, and detailed enough to give rise to a non-trivial reward machine) safe abstraction is not guaranteed. However, generating a useful safe abstraction is not

⁴If the sets of propositional symbols of the MGRM and the ECGS are not the same, then s and q must satisfy the same propositional symbols up to the function that connects the two sets.

a more demanding task than generating an abstract planning domain corresponding to an MDP as in (Kokel et al., 2021) or a set of high-level options implementable by RL policies as in (Sutton, Precup et al., 1999), given the similarities of these approaches with our notion of safe abstractions.

5.3 Synthesising Markov Games With Reward Machines

In this section, we show how, given a multi-agent environment E , a labelling L for it, and an FC-ATL formula $\langle\langle \text{Agt} \rangle\rangle\varphi$ specifying the team task, we can synthesise a MGRM for any set of agents $B \subseteq \text{Agt}$ (MDPRM for singletons of agents) from a witness for $\langle\langle \text{Agt} \rangle\rangle\varphi$. While for generality B may be any subset of Agt , we focus on the cases where $B = \text{Agt}$ and where B is a singleton, i.e., on generating the coalition MGRM and the MDPRM for each agent. The aim of this construction is to obtain the reward machine for the set of agents B . If B is the grand coalition, the MGRM can be obtained by just “appending” the reward machine to the (labelled) multi-agent environment that is given in input. If B is a singleton $\{i\}$, its MDPRM is obtained by deriving the labelled single-agent environment E_i of agent i from the multi-agent environment and then appending the reward machine to it. In Figure 5.4, we provide a high-level overview of the objects that are used in our approach and how they are related to each other. The MAE, the ECGS and the FC-ATL formula are all given as input, whereas the rest is computed in our approach. We would like to stress the dynamics of the low-level environments (MAE, SAE, MGRM, and MDPRM) are hidden from the agents, which means that it is not possible for them to compute a policy to perform the task given only a witness of a high-level strategy (which is obtained in the ECGS) for it. Finally, we observe that so long as the FC-ATL formula is true given the ECGS and the initial ECGS state, we can generate a witness and thus the reward machines. In case the FC-ATL formula is false, then it is not possible to carry out our synthesis procedure.

5.3.1 Synthesising Reward Machines (via a witness)

Fix a multi-agent environment $E = \langle \text{Agt}, S_1, \dots, S_n, A_1, \dots, A_n, T \rangle$. Given an ECGS $M = \langle \text{Agt}, Q, (Prop_i)_{i \in \text{Agt}}, V, (\sim_i)_{i \in \text{Agt}}, Act, d, \delta \rangle$ which is safe for E , we can use an ATL model checker to synthesise a witness of a uniform strategy that achieve the task encoded by the FC-ATL formula $\langle\langle A \rangle\rangle\varphi$. From the witness, we can synthesise both the coalition RM $\mathcal{R}^A$ and each of the individual RMs $\mathcal{R}^i$, for each agent $i \in A$. Notice that the witness and the reward machine derived from it are defined in terms of high-level actions from the ECGS M and are not directly executable in the MAE E (or in the individual agents’ environments). However, the synthesised RM can be used to guide an agent in learning which low-level actions in E should be performed to accomplish each step in the RM.

The construction of a reward machine from a witness $W(q, \Pi_A)$ is given in Construction 5.1. First we explain the construction informally. The set of states of the coalition RM is the same set of the witness, whereas the set of states of agent i ’s individual RM is the set of equivalence classes, with respect to the indistinguishability

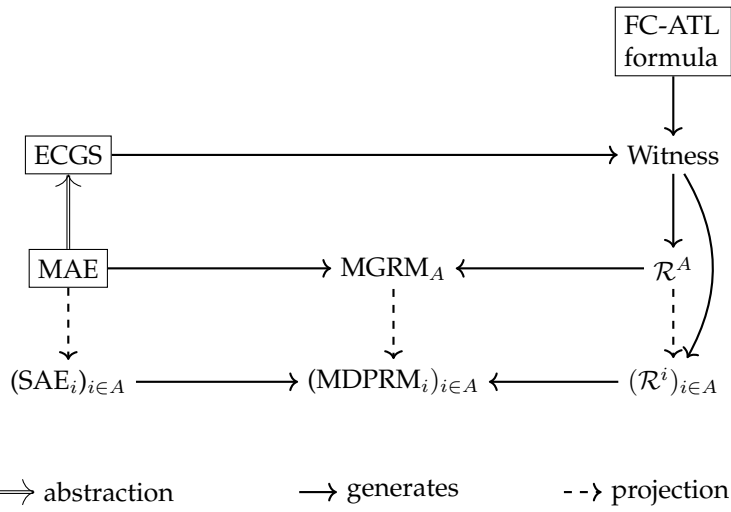


Figure 5.4: Overview and relationship between the objects used in our approach. Objects in a box are given as input by the user, who we assume has access to an ECGS which is a safe abstraction of the MAE.

relation $\sim_i$, of the states in the witness. In both cases, we include an extra “error” state. For the edge labels, we simply replace the high-level actions that label such edges in the original witness with their corresponding postconditions. For the coalition RM, transition between states are the same as in the witness. Instead, for an individual agent’s RM, transitions between states, i.e., equivalence classes, are defined depending on the transitions between states in such equivalence classes. Given two equivalence classes, if there is a state in the former that transitions to a state in the latter given some postcondition, then there will be a transition between the two classes labelled with the same postcondition. The reward machines for $\langle\langle A \rangle\rangle\phi U \psi$ and $\langle\langle A \rangle\rangle G \phi$ transit to the error state on events corresponding to a violation of the invariant property ϕ (we assume that $\neg\phi$ is always observable by any agent in A). The error state has a self-loop and no transitions to other states of the reward machine. As a minor complication, the state corresponding to the second last state of the witness for $\langle\langle A \rangle\rangle\phi U \psi$ transits to the error state only on events that *both* violate ϕ *and* are not postconditions of the last action in the witness (i.e., do not achieve ψ).

Notice that, although the construction can be applied to any subcoalition $B \subseteq A$, we are only interested in generating the RMs for the entire coalition A and for each individual agent $i \in A$.

Construction 5.1 (Construction of a reward machine). Given the ECGS $M = \langle \text{Agt}, Q, (\text{Prop}_i)_{i \in \text{Agt}}, V, (\sim_i)_{i \in \text{Agt}}, \text{Act}, d, \delta \rangle$ and a witness $W(q, \Pi_A)$, we construct a reward machine $\mathcal{R}^B = \langle U^B, u_0^B, F^B, \Sigma^B, \delta_u^B, \delta_r^B \rangle$ for $B \subseteq A$ as follows:

- $U^B := Q(W(q, \Pi_A)) / \sim_B \setminus Q_f(W(q, \Pi_A)) / \sim_B$ is the set of RM states, defined as the set of equivalence classes of states in $W(q, \Pi_A)$ with respect to the indistinguishability relation of B but excluding the set of equivalence classes of final states in $W(q, \Pi_A)$ with respect to the indistinguishability relation of B (the latter will be the set of final states of the RM);
- $u_0^B := [q]_B$;
- $F^B := Q_f(W(q, \Pi_A)) / \sim_B \cup \{u_{err}\}$ is the set of final states, defined as the set of equivalence classes of final states in $W(q, \Pi_A)$ with respect to the indistinguishability relation of B , plus the error state u_{err} ;
- $\Sigma^B := 2^{\bigcup_{i \in B} \overline{\text{Prop}_i}}$. If $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle G \phi$ or $\langle\langle A \rangle\rangle\phi U \psi$, then we also add all the events e that entail $\neg\phi$ to Σ^B ;
- $\delta_u^B(u, e) := u'$ if and only if there are q_1, q_2 in the set of nodes of $W(q, \Pi_A)$ such that $u = [q_1]_B$ and $u' = [q_2]_B$ with q_1 connected to q_2 through an edge labelled with $\Pi_A(q_1)$, and there is a joint action $\sigma \in D(q_1)$ with $q_1 \xrightarrow{\sigma} q_2$, $\sigma_A = \Pi_A(q_1)$ such that $e = \overline{\text{post}(\sigma)} \upharpoonright \Sigma^B$. If $\langle\langle A \rangle\rangle\varphi = \langle\langle A \rangle\rangle G \phi$ or $\langle\langle A \rangle\rangle\varphi = \langle\langle A \rangle\rangle\phi U \psi$ and $e \models \neg\phi$ then $u' = u_{err}$ unless $e = \overline{\text{post}(\sigma)} \upharpoonright \Sigma^B$ for an action σ leading to a final state q_f in $W(q, \Pi_A)$. In the latter case the RM transitions to $u' = [q_f]$ in the witness $W(q, \Pi_A)$ (as ϕ needs to hold only strictly before ψ holds);
- For the definition of δ_r^B , there are three different cases:

1. $\delta_r^B(u, e) := 1$ if and only if $\langle\langle A \rangle\rangle\varphi = \langle\langle A \rangle\rangle X\psi$, $u = [q]$, and $e = \overline{\text{post}(\Pi_A(q))} \upharpoonright \Sigma^B$, or $\langle\langle A \rangle\rangle\varphi = \langle\langle A \rangle\rangle\phi U\psi$ and $e = \text{post}(\sigma) \upharpoonright \Sigma^B$ for an action σ leading to a final state q_f in $W(q, \Pi_A)$.
2. $\delta_r^B(u, e) := -1$ if and only if $\langle\langle A \rangle\rangle\varphi = \langle\langle A \rangle\rangle G\phi$, $u \neq u_{err}$ and e entails $\neg\phi$;
3. For all other⁵ (u, e) , $\delta_r^B(u, e) := 0$.

We now show that the RMs synthesised by our approach are “correct” decompositions in the sense of Neary et al. (2021), i.e., that under certain sequences of events, both the individual and coalition RMs will give the same reward. First, we define the product of RMs.

Definition 5.11 (Product of RMs). Fix RMs for agents $i_1, \dots, i_n$, where the RM for agent i_k is $\mathcal{R}^k = \langle U^k, u_0^k, F^k, \Sigma^k, \delta_u^k, \delta_r^k \rangle$, based on a common witness $W(q, \Pi_A)$ with the event sets Σ^k obtained from the set of observables Prop_{i_k} . We define the product RM $\mathcal{R}^\otimes = \langle U^\otimes, u_0^\otimes, F^\otimes, \Sigma^\otimes, \delta_u^\otimes, \delta_r^\otimes \rangle$ of the RMs, denoted by $R_1 \otimes \dots \otimes R_n$ as follows:

- $U^\otimes := U^1 \times \dots \times U^n$;
- $u_0^\otimes := (u_0^1, \dots, u_0^n)$;
- $F^\otimes := (F^1 \setminus \{u_{err}\}) \times \dots \times (F^n \setminus \{u_{err}\}) \cup \{u_{err}\}$;
- $\Sigma^\otimes$ is the event set obtained from the set of observables $\text{Prop}_{i_1} \cup \dots \cup \text{Prop}_{i_n}$;
- $\delta_u^\otimes((u_1, \dots, u_n), e) := (\delta_u^1(u_1, e \upharpoonright \Sigma^1), \dots, \delta_u^n(u_n, e \upharpoonright \Sigma^n))$. If $\langle\langle A \rangle\rangle\varphi = \langle\langle A \rangle\rangle G\phi$ or $\langle\langle A \rangle\rangle\varphi = \langle\langle A \rangle\rangle\phi U\psi$ and e entails $\neg\phi$ then $\delta_u^\otimes((u_1, \dots, u_n), e) = u_{err}$;
- For the definition of $\delta_r^\otimes$, we have again three cases:
 1. $\delta_r^\otimes((u_1, \dots, u_n), e) := 1$ if $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle X\psi$ or $\langle\langle A \rangle\rangle\phi U\psi$ and $\delta_u^\otimes((u_1, \dots, u_n), e) \in F^\otimes \setminus \{u_{err}\}$;
 2. $\delta_r^\otimes((u_1, \dots, u_n), e) := -1$ if $\langle\langle A \rangle\rangle\varphi = \langle\langle A \rangle\rangle G\psi$ and e entails $\neg\psi$;
 3. $\delta_r^\otimes((u_1, \dots, u_n), e) := 0$ otherwise.

Let A be a coalition of agents, Π_A a strategy for the coalition and q the initial ECGS state. We define the set of *compatible event sequences* $\Xi_{\Pi_A, q}$ for strategy Π_A and initial state q as the set of event sequences that is observed by coalition A while following strategy Π_A , i.e. $\Xi_{\Pi_A, q} := \{\xi \mid \exists \lambda \in \text{out}(q, \Pi_A) \forall i \in \mathbb{N} \exists \sigma \in D(\lambda[i]) : \sigma_A = \Pi_A(\lambda[i]) \wedge \xi[i] = \text{post}(\sigma) \upharpoonright \Sigma^A\}$.

Finally, before formalizing and proving our ‘correctness’ claim, we state and prove a lemma about the states of the coalition RM and of the product RM. Fix a coalition of agents $A = \{i_1, \dots, i_n\}$, its coalition RM $\mathcal{R}^A = \langle U^A, u_0^A, F^A, \Sigma^A, \delta_u^A, \delta_r^A \rangle$ and a set of observables Σ^A for the coalition. For each agent $i \in A$, let $\mathcal{R}^i = \langle U^i, u_0^i, F^i, \Sigma^i, \delta_u^i, \delta_r^i \rangle$ be its individual RM, with $\Sigma^i \subseteq \Sigma^A$ be the set of observables of such agent.

⁵Note that, in case the task is specified by a formula of the form $\langle\langle A \rangle\rangle\phi U\psi$ and the formula ϕ is violated, the agents in B are given a reward of 0, while they receive a reward of -1 when the task is encoded by a formula of the form $\langle\langle A \rangle\rangle G\phi$. While counterintuitive, this is crucial in order to prove Theorem 5.3 below. Note that, even though the agents do not receive a negative reward for violating the invariant ϕ , it is still the case that their optimal (joint) policy must achieve the task: the only way for them to receive a non-zero reward is to satisfy ψ while maintaining the invariant ϕ true.

Lemma 5.1. *There is a (partial) homomorphism g , with respect to the transition functions of the RMs, from states and final states of the product RM $\mathcal{R}^\otimes$ to those of the coalition RM $\mathcal{R}^A$.*

Proof. We prove the statement only for the set of non-final states of the RMs, as the argument for the sets of final states is identical. For the coalition RM, states are equivalence classes of $\sim_A$, whereas in the product RM they are tuples of equivalence classes, containing precisely one class for each agent $i \in A$. We show that any state $u \in U^A$ can be obtained by some other state $u' \in U^\otimes$. This mapping is induced by observing that an equivalence class of $\sim_A$ is the intersection of a tuple containing one equivalence class per $\sim_i$. To prove this, we show that for an arbitrary state $q \in Q$, we have that $[q]_A = \bigcap_{i \in A} [q]_i$. First, we show that $[q]_A \subseteq \bigcap_{i \in A} [q]_i$: let q' be an arbitrary $q' \in [q]_A$. Since $q' \in [q]_A$ we have that q' agrees with q on the propositions observable by A . Notice that the set of observable propositions $Prop_A$ is exactly the union of the set of observable propositions $\bigcup_{i \in A} Prop_i$. Thus, let $i \in A$ be arbitrary: it is easy to see that $q' \in [q]_i$, because as q' agrees on the propositional symbols in $Prop_A$ with q , it must also agree on the propositional symbols in $Prop_i$ with q , since $Prop_i \subseteq Prop_A$. Thus, $q' \in \bigcap_{i \in A} [q]_i$. For the other inclusion, the proof is similar: suppose that $q' \in \bigcap_{i \in A} [q]_i$. Then, it must agree with q on the propositional symbols in $\bigcup_{i \in A} Prop_i$. Since $Prop_A = \bigcup_{i \in A} Prop_i$, it follows that $q' \in [q]_A$. We denote by $g : U^\otimes \rightharpoonup U^A$ the partial function such that $g((u^{i_1}, \dots, u^{i_n})) := \bigcap_{i \in A} u_i$ for any $(u^{i_1}, \dots, u^{i_n}) \in U^{i_1} \times \dots \times U^{i_n}$, if such intersection is non-empty, and is otherwise undefined.

We can now prove that g is a homomorphism between $U^\otimes$ and U^A with respect to the transition functions $\delta_u^\otimes$ and δ_u^A . We need to show that $g(\delta_u^\otimes(u_0^\otimes, \xi[j])) = \delta_u^A(g(u_0^\otimes), \xi[j])$ for any index j in an arbitrary event sequence ξ . We prove this claim by induction on the number of steps j . We first consider the base case, i.e., $j = 0$. Notice that at $j = 0$, $u_0 = u_0^\otimes$, thus $g(u_0) = u_0^A$. After observing event $\xi[0]$, the product RM moves to the new state $u_1 = (u_1^{i_1}, \dots, u_1^{i_n})$, where each u_1^i is the state to which the RM of agent i transitions from u_0^i after observing $\xi[0]$, while the coalition RM moves to the new state u_1^A . We want to show that $g(u_1) = u_1^A$. Since the RM of agent i moves according to the observable propositions in $Prop_i$, notice that the new RM state u_1^i is the equivalence class (with respect to $\sim_i$) of the states of the ECGS which: (i) agree with the states in the equivalence class represented by the RM state u_0^i on propositional symbols in $Prop_i$ for which no literal appears in $\xi[0]$, and (ii) have the truth values of propositional symbols in $Prop_i$ corresponding to the literals appearing in $\xi[0]$ (i.e., if $p^+ \in \xi[0]$, then p must be true in the states in u_1^i , and if $p^- \in \xi[0]$, then p must be false in such states). By taking the intersection of all the new states of the individual RMs, it is clear that the previous points apply to the whole set of observables $Prop_A$. Also, the intersection cannot be empty as two agents cannot observe contradictory literals. Now, notice that the coalition RM transitions in the same way as the RM of any agent i , but considering $Prop_A$ instead of $Prop_i$. Therefore, we have indeed that $g(u_1) = u_1^A$. For the inductive case $j = n + 1$, the argument is exactly the same, albeit we need to apply the inductive hypothesis to know that $g(u_n) = u_n^A$. $\square$

Now, we can give the claim and prove it.

Theorem 5.2. *Fix a strategy Π_A for the formula $\langle\langle A \rangle\rangle\varphi$ for the coalition of agents $A = \{i_1, \dots, i_n\}$ and an initial ECGS state q . Let $\mathcal{R}^A = \langle U^A, u_0^A, F^A, \Sigma^A, \delta_u^A, \delta_r^A \rangle$ be the coalition RM and $\mathcal{R}^\otimes = \langle U^\otimes, u_0^\otimes, F^\otimes, \Sigma^\otimes, \delta_u^\otimes, \delta_r^\otimes \rangle$ be the product RM $\mathcal{R}^{i_1} \times \dots \times \mathcal{R}^{i_n}$. Given a compatible event sequence $\xi \in \Xi_{\Pi_A, q}$, then for any step $j \in |\xi|$ we have that $\delta_r^\otimes(u_j, \xi[j]) = \delta_r^A(g(u_j), \xi[j])$, where $u_j = (u_j^{i_1}, \dots, u_j^{i_n})$ is the j -th state reached by the product RM following the event sequence ξ and $g : U^\otimes \rightarrow U^A$ is defined as in the proof of Lemma 5.1.*

Proof. The proof essentially amounts to showing that g not only preserves transitions, but also output rewards. We consider two different cases, depending on the structure of the formula that encodes the task. If $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle X\psi$ or $\langle\langle A \rangle\rangle \phi U\psi$, then $\mathcal{R}^\otimes$ can either output 0 or 1 at any step j . In case it outputs 0, either some agent has not reached one of its achievement states in its RM or the invariant ϕ , for $\langle\langle A \rangle\rangle \phi U\psi$ formulas, has been falsified (notice in this latter case how every agent is able to observe that ϕ has become false). Thus, let u_j be the state of the product RM $\mathcal{R}^\otimes$ at step j and let $\xi[j]$ be the event observed at such step. By Lemma 5.1, we know that u_j^A , the state of the coalition RM $\mathcal{R}^A$ at step j , is exactly $g(u_j)$. Thus, assume by way of contradiction that $\delta_r^A(u_j^A, \xi[j]) \neq 0$: this can only mean that $\delta_r^A(u_j^A, \xi[j]) = 1$, i.e., that the whole coalition is able to observe that ψ has become true. However, this contradicts our starting assumption, thus showing the claim. Instead, if $\mathcal{R}^\otimes$ outputs 1, then this must be due to the fact that from state u_j and thanks to event $\xi[j]$ all agents in A will be able to observe that the following state will be a goal one, i.e., a state where ψ is true. Hence, their respective RMs will all be in one of the final states. From here, we can prove the claim with a similar argument to the one we gave for the other case.

Finally, if $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle G\psi$, then we can show the claim by considering the same argument we gave before in case the reward was 0. Recall that for these formulas, the RMs give a reward of 0 so long as the invariant is true, and -1 when it is falsified. Thus, it suffices to consider exactly that argument, as it already covers such two cases for formulas of the form $\langle\langle A \rangle\rangle \phi U\psi$. $\square$

Theorem 5.2 thus establishes that, with respect to output rewards, the coalition RM and the RM obtained from the product of all the individual RMs are indistinguishable. Therefore, we immediately obtain a result analogous to Theorem 1 from Neary et al. (2021), without the need to check whether such RMs are bisimilar.

Note that in Theorem 5.2 we consider the case in which the RM initial state is fixed. To construct a reward machine that works from an arbitrary initial state, we construct a reward machine $W(q, \Pi_A)$ for every $q \in Q$ and then connect them by adding an extra state u_0 that has a transition labelled by $e \in \Sigma^A$ where e describes each equivalence class with respect to $\sim_A$. Since the truth definition for $\langle\langle A \rangle\rangle\varphi$ we use corresponds to strong uniformity, there is a uniform strategy F_A that works from all states indistinguishable by $\sim_A$, and all different classes are distinguishable by some event $e \in \Sigma^A$. The proof of Theorem 5.2 carries over to the case when the initial state is not fixed.

To conclude, we state a theorem relating the future undiscounted rewards obtained by a coalition and the individual rewards obtained by the agents in the coalition.

Consider a coalition $A = \{i_1, \dots, i_n\}$, a witness $W(q, \langle\langle A \rangle\rangle\varphi)$ for some formula $\langle\langle A \rangle\rangle\varphi$ from state q of an ECGS, a joint state s of a MAE and an arbitrary joint policy $\pi = (\pi_1, \dots, \pi_n)$ for the same MAE. For the coalition RM $\mathcal{R}^A$ built from $W(q, \langle\langle A \rangle\rangle\varphi)$, we denote by $v_\pi^A(s)$ the sum of expected future undiscounted rewards produced by $\mathcal{R}^A$, given all agents follow, from state s of the MGRM and the initial state u_0^A of $\mathcal{R}^A$, their policy as specified by π . Analogously, for the individual RM $\mathcal{R}^{i_j}$ built from the same witness $W(q, \langle\langle A \rangle\rangle\varphi)$, we denote by $v_\pi^{i_k}(s)$ the sum of expected future undiscounted rewards produced by $\mathcal{R}^{i_j}$ under the same assumptions. Moreover, recall that if $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle X\psi$ or $\langle\langle A \rangle\rangle \phi U \psi$, then any RM synthesised from a witness for a strategy for the formula can give a reward of only 0 or 1. Similarly, any RM for a formula of the form $\langle\langle A \rangle\rangle G\phi$ can give a reward of only 0 or -1 .

From the observations above and the definition of $v_\pi^B(s)$, regardless of the policy π , the environment state s and of whether we are considering a coalition or a single agent, it follows that if the formula is of the form $\langle\langle A \rangle\rangle X\psi$ or $\langle\langle A \rangle\rangle \phi U \psi$, then $v_\pi^B(s)$ is exactly the probability that the (sub)coalition $B \subseteq A$ is able to complete the (sub)task encoded by their RM. On the other hand, if the formula is of the form $\langle\langle A \rangle\rangle G\phi$, then $-v_\pi^B(s)$ is equal to the probability of failing to complete the (sub)task.

Theorem 5.3. *If $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle X\psi$ or $\langle\langle A \rangle\rangle \phi U \psi$, then*

$$\max\{0, v_\pi^{i_1}(s) + \dots + v_\pi^{i_n}(s) - (n-1)\} \leq v_\pi^A(s) \leq \min\{v_\pi^{i_1}(s), \dots, v_\pi^{i_n}(s)\}$$

If $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle G\psi$, then

$$\max\{-v_\pi^{i_1}(s), \dots, -V_{i_n}^\pi(s)\} \leq -v_\pi^A(s) \leq \min\{1, -v_\pi^{i_1}(s) - \dots - v_\pi^{i_n}(s)\}.$$

Proof. If $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle X\psi$ or $\langle\langle A \rangle\rangle \phi U \psi$, then notice all RMs will give a reward of 1 when the task they encode has been completed and 0 otherwise. Thus, we can see the expected sum of undiscounted future rewards $v_\pi^A(s)$ as the probability that coalition A completes the task assuming every agent is following the joint policy π . Notice that the same holds as well for any other agent $i_k \in A$. Moreover, by Theorem 5.2 we know that for any compatible event sequence $\xi \in \Xi_{\Pi_A, q}$ at any timestep the rewards given by the individual RMs and the coalition one are equal. Thus, the probability that the whole coalition A completes the task described by $\mathcal{R}^A$ is equal to the probability that all agents complete the individual tasks described by their respective RMs, assuming that all agents follow the joint policy π . Then, by applying directly the Frechét inequality for logical conjunctions we get the first part of the claim.

On the other hand, if $\langle\langle A \rangle\rangle\varphi$ is of the form $\langle\langle A \rangle\rangle G\psi$, it suffices to notice that the opposite of the expected sum of undiscounted future rewards (both for coalition A and for any agent $i_k \in A$) is the probability of failing the task. This is because in this case the RMs will output -1 when the task they describe is failed and 0 otherwise. Now, observe that as soon as one agent fails the task, then all other agents will fail the task as the invariant has been falsified. Since every agent can observe whether the invariant has been falsified, their RMs will move to the error state and give a reward

of -1 . By Theorem 5.2 then also the coalition RM will give a reward of -1 at such timestep. On the other hand, if the coalition RM gives a reward of -1 , then all the individual RMs will also give a reward of -1 , by the same reasoning. Therefore, the probability that coalition A fails the task is equivalent to the probability that at least a single agent in A fails the task. Thus, by applying the Frechét inequalities for logical disjunctions we get the second part of the claim. $\square$

Intuitively, Theorem 5.3 bounds the probability that a coalition completes a task (or not, depending on the formula type), assuming all agents follow their policy specified by π . For formulas of the form $\langle\langle A \rangle\rangle X\psi$ or $\langle\langle A \rangle\rangle \phi \cup \psi$, if all agents $i \in A$ are able to (eventually) complete their subtask, then coalition A is able to (eventually) complete the joint task: if all the individual $v_{\pi}^{i_k}(s) = 1$, then $\max\{0, v_{\pi}^{i_1}(s) + \dots + v_{\pi}^{i_n}(s) - (n - 1)\} = 1$ and so $v_{\pi}^A(s) = 1$. Similarly, for formulas of the form $\langle\langle A \rangle\rangle G\psi$. If no agent violates ψ , then the coalition will not violate ψ : if each individual $-v_{\pi}^{i_k}(s) = 0$, then $\min\{1, -v_{\pi}^{i_1}(s) - \dots - v_{\pi}^{i_n}(s)\} = 0$ and so $v_{\pi}^A(s) = 0$.

Theorem 5.3 is analogous to Theorem 2 from Neary et al. (2021), but our claim is made with respect to FC-ATL formulae, instead of RMs. Both results establish the same idea, i.e., that it makes no difference to the probability of task completion whether agents are trained individually or together.

5.4 Evaluation

In this section we present an evaluation of automatically synthesised reward machines. Specifically, we show how strategies for team tasks generated by the model checker MCMAS (Lomuscio, Qu et al., 2017) can be used to produce an RM for each agent in a team of agents. We present results in four benchmarks: COOPERATIVEBUTTONS and RENDEZVOUS from (Neary et al., 2021), OFFICEWORLD from (Toro Icarte, Klassen, Valenzano and McIlraith, 2022), and WATERWORLD from (Karpathy, 2015).⁶

For all experiments, we present the results in plots where lines represent median performance, and the shaded areas show the 25th and 75th percentiles. For the experiments comparing against the RMs of Neary et al. (2021) and the ones in WATERWORLD, performance is measured as the number of steps necessary to complete the task, whereas in the experiment where we compare against the RM of Toro Icarte, Klassen, Valenzano and McIlraith (2022) the performance measure is the average reward per step, as in their experiments.

5.4.1 Synthesising Reward Machines

Before presenting the experimental results we motivate the use MCMAS, the model checker we have used to obtain witnesses from FC-ATL formulas, and briefly discuss alternative tools which could have been used.

⁶Our code can be found at <https://github.com/giovannivarr/MARM-JAIR-code>; the code of Neary et al. (2021) at github.com/cyrusneary/rm-cooperative-marl, and the code of Toro Icarte, Klassen, Valenzano and McIlraith (2022) at github.com/RodrigoToroIcarte/reward_machines.

MCMAS⁷ is a model checker which is able to produce witnesses for formulas in several logics, including ATLK and CTLK (Lomuscio, Qu et al., 2017). This ability to model imperfect information scenarios and to generate uniform strategies is crucial for our approach. The generation of uniform strategies (and witnesses) by MCMAS is enabled using `-uniform` flag. However, it is important to note that the ATL semantics used by MCMAS with the `-uniform` flag differs from the standard semantics for ATL with imperfect information and memoryless strategies. When the `-uniform` flag is enabled, MCMAS first creates all possible *uniform protocols* for each agent (Lomuscio and Raimondi, 2006). A protocol is uniform if, in all states indistinguishable by an agent, the agent is restricted to a single action from the set of actions available in that state for the rest of the run. For each set of uniform protocols for each agent in the coalition, MCMAS generates a uniform strategy, and checks whether the *entire* ATL formula is satisfied under this strategy, i.e., the same uniform strategy is used for all modalities in the formula. If the formula is satisfied, MCMAS halts and outputs true; otherwise it moves to the next uniform strategy. If no uniform coalition strategy (i.e., none of the possible profiles containing a uniform protocol per each agent) satisfy the formula, MCMAS outputs false. This differs from the standard uniform semantics for ATL, where each nested coalitional modality may be associated with a different (uniform) strategy. Nevertheless, the following observation is, as we will see, almost immediate.

Observation 5.1. With MCMAS’s `-uniform` flag enabled, the semantics of FC-ATL and the standard semantics of ATL with imperfect information and memoryless strategies are equivalent.

This follows from the fact that in FC-ATL formulas contain exactly one coalitional modality for the grand coalition. Thus FC-ATL formulas are true if and only if there is a single uniform strategy for the grand coalition which makes the entire formula true. Hence, using MCMAS with the `-uniform` flag enabled will correctly produce a witness for a uniform strategy for an FC-ATL formula encoding a coalition’s task.

We conclude this section with a brief discussion of alternative tools. STV (Kurpiewski et al., 2024) and VITAMIN (Ferrando and Malvone, 2025) are model checkers which support model checking of ATL formulas in a setting with imperfect information. However, VITAMIN does not currently support witness generation, and at the time when the work on this paper started (2023), the actively maintained STV version supported only singleton coalitions. MCK is a model checker which is able to check formulas in several temporal logics (including CTL and LTL) and which also provides support for imperfect information through the inclusion of knowledge modalities (Gammie and Meyden, 2004). Moreover, MCK can produce counterexamples for LTL formulas of the form $\bigcirc\phi$ and $\Box\phi$ (constrained to formulas where ϕ can contain knowledge modalities but not temporal ones). By synthesising a counterexample for LTL formulas of the form $\bigcirc\neg\phi$ and $\Box\neg\phi$, MCK can be used to obtain witnesses for tasks encoded by ATL formulas of the form $\langle\langle A \rangle\rangle X\psi$ (where $\psi \equiv \neg\psi$) and $\langle\langle A \rangle\rangle T U\phi$. However, it is not possible to obtain a witness for formulas of the form $\langle\langle A \rangle\rangle G\phi$ or $\langle\langle A \rangle\rangle\phi U\psi$ using MCK, as these formulas are not equivalent to any formula for which

⁷We used MCMAS version 1.3.0 to generate the witnesses in the experiments below.

MCK can produce a counterexample. As such, MCK can only generate witnesses for a fragment of FC-ATL. There are also several tools for CTL/CTL* synthesis. For CTL, these include (Klenze et al., 2016), (De Angelis et al., 2012), and CTLSAT, an implementation of the original CTL synthesis algorithm proposed in (Clarke and Emerson, 1981),⁸ and for CTL* synthesis (Bloem, Schewe et al., 2017), (Khalimov and Bloem, 2017), and (Khalimov, 2018). Finally, both Strix (Meyer et al., 2018; Luttenberger et al., 2020) and Syft (S. Zhu et al., 2017) can synthesise strategies for LTL/LTL_f specifications. However, all these synthesis tools lack direct support for incomplete information and the actions of individual agents without significant modification or extension. In (Tabajara and Vardi, 2020) an approach to LTL synthesis under partial information is presented; however, to the best of our knowledge, no implementation is publicly available. In summary, MCMAS provides us with exactly what we need: a way to generate strategies for tasks encoded in FC-ATL and accounting for partial information. Nevertheless, we stress that our approach is agnostic to the tool used to model check FC-ATL.

5.4.2 COOPERATIVEBUTTONS & RENDEZVOUS

We first compared the performance of DQPRM when using our automatically synthesised RMs and the RMs by Neary et al. (2021) in both the COOPERATIVEBUTTONS and RENDEZVOUS environments. The COOPERATIVEBUTTONS environment was introduced in Section 5.2 (Figure 5.1). The RENDEZVOUS environment is a 10x10 gridworld in which the task of the agents is to first rendezvous at a common location and then for each agent to reach their own goal. For the RENDEZVOUS task we present results for teams of 10 and 25 agents. The 10-agent case allows comparison with the same experiment performed by Neary et al. (2021). The 25-agent case was performed to determine the scalability of our approach, from the strategy generation using MCMAS to the agents' training using DQPRM.

We used the same experimental setup of Neary et al. (2021) for these two tasks. For action selection, agents used softmax exploration with a constant temperature of $\tau = 0.2$. The discount factor was $\gamma = 0.9$ and the learning rate $\alpha = 0.8$. As in (Neary et al., 2021), for both tasks each experiment consisted of 10 episodes. For the COOPERATIVEBUTTONS task each episode consisted of 250,000 training steps, while for the RENDEZVOUS task each episode consisted of 150,000 training steps. For both tasks a test was run every 1,000 training steps to evaluate the agents, with every test running for at most 1,000 steps (if the task was not achieved in 1,000 steps it was considered failed and the agent received no reward). It was necessary to slightly modify the original code used by Neary et al. (2021) to work with our automatically synthesised RMs. We note that this did not affect the results for the Neary et al. (2021) RM, as the code we introduced was used only when training agents with our RMs. Both the COOPERATIVEBUTTONS and RENDEZVOUS experiments were run on a machine with an Apple M1 CPU and 16 GB RAM.

The results are shown in Figures 5.5, 5.6, and 5.7. As it can be seen, in the COOPERATIVEBUTTONS and 10-agents RENDEZVOUS tasks, the performance of our

⁸CTLSAT's implementation can be found at <https://github.com/nicolaprezza/CTLSAT>.

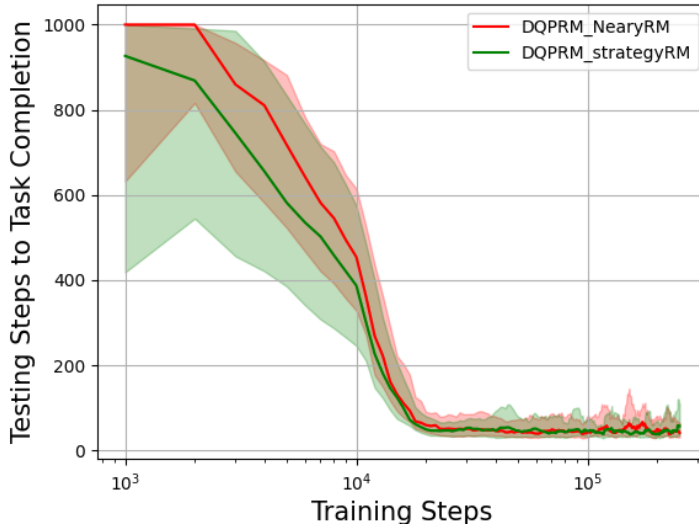


Figure 5.5: Three-agents COOPERATIVEBUTTONS task (Neary et al., 2021). Timeout is set to 1,000 timesteps, at which point the task is considered failed.

automatically synthesised RMs is comparable to that of the RMs of Neary et al. (2021) in terms of sample efficiency, suggesting that, at least for these scenarios, automatically synthesized RMs allow the agents to learn the task. The results for the 25 agent RENDEZVOUS task are shown in Figure 5.7. While the agents are still able to learn to achieve the task in a number of steps similar to the one required to achieve the task by 10 agents (as learning is decentralised), generation of the coalition RM can be a bottleneck, as witness generation using MCMAS required 23 hours. However, we note that approaches in the cooperative MARL literature usually do not consider larger numbers of agents. For example, a recent survey by Cui et al. (2022) notes that the number of agents in cooperative MARL typically does not exceed 10 (see, e.g., (Papoudakis et al., 2021)). An exception is the work of Ellis et al. (2023) that benchmarks cooperative MARL algorithms with coalitions of up to 23 agents in StarCraft domains.

5.4.3 OFFICEWORLD

In the OFFICEWORLD scenario, we compare our automatically synthesised RM to that of Toro Icarte, Klassen, Valenzano and McIlraith (2022) for the same task. The OFFICEWORLD environment is a 12x9 gridworld (see Figure 5.8), in which the objective is to collect coffee and mail and bring it to the office. The environment contains some decorations which the agent must not break while performing the task. Thus, the agent must learn how to navigate the world in order to collect the resources and then bring them to the office while avoiding decorations. We present results from

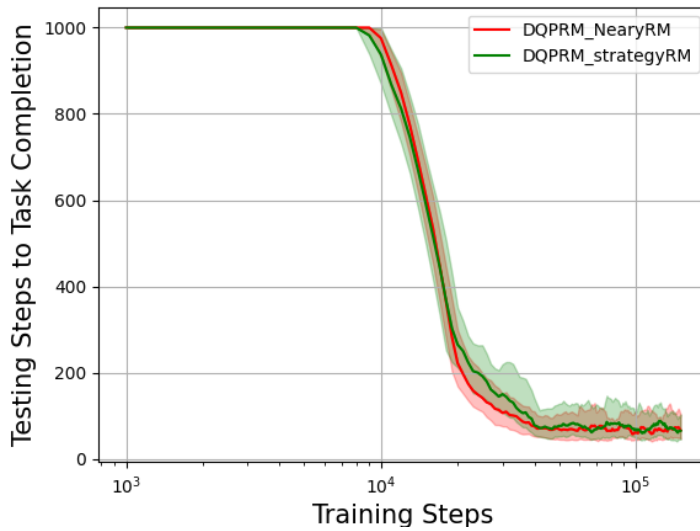


Figure 5.6: Ten-agents RENDEZVOUS task (Neary et al., 2021). Timeout is set to 1,000 timesteps, at which point the task is considered failed.

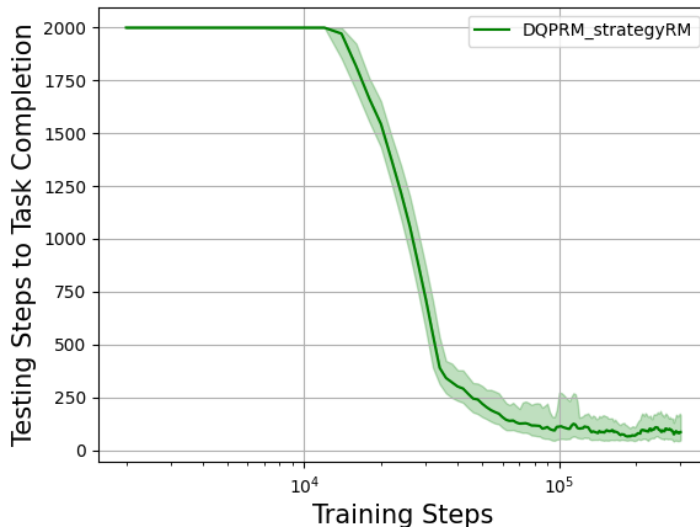


Figure 5.7: Twenty-five-agents RENDEZVOUS task (Neary et al., 2021). Timeout is set to 2,000 timesteps, at which point the task is considered failed.

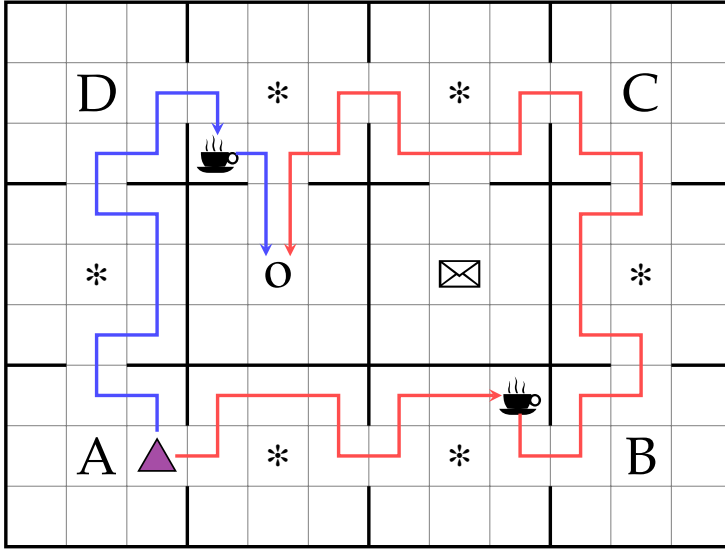


Figure 5.8: OFFICEWORLD rendition from (Toro Icarte, Klassen, Valenzano and McIlraith, 2022).

this environment to: (i) show how our approach can also handle tasks where an invariant must be maintained, and (ii), how it can be used to synthesise RMs also in the single-agent case.

We used the same experimental setup as the one of Toro Icarte, Klassen, Valenzano and McIlraith (2022), i.e., the exploration rate $\epsilon = 0.1$, the discount factor $\gamma = 0.9$, and the learning rate $\alpha = 0.5$. We trained the agents using two algorithms introduced by Toro Icarte, Klassen, Valenzano and McIlraith (2022): the cross-product baseline (QL) and the algorithm exploiting counterfactual experiences (CRM). Agents were trained for 60 episodes, each lasting 100,000 training steps. In the OfficeWorld scenario, we were able to use the code and RM of Toro Icarte, Klassen, Valenzano and McIlraith (2022) without modification, by simply adding a .txt file encoding our automatically synthesised RM. Due to compatibility requirements with the code of Toro Icarte, Klassen, Valenzano and McIlraith (2022), the OFFICEWORLD experiment was run on a machine equipped with a 2.3 GHz quad-core Intel Core i5 CPU and an 8 GB RAM.

The results are shown in Figures 5.9 and 5.11. As it can be seen, the performance of our automatically synthesised RMs is again comparable to that of the RM used by Toro Icarte, Klassen, Valenzano and McIlraith (2022). Indeed, the agent trained with our automatically synthesised RM converges faster to the optimal reward per step compared to the agent trained with the RM of Toro Icarte, Klassen, Valenzano and McIlraith. We believe this is due to the synthesised RM being a “sub-RM” of the RM in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022) (see Figure 5.10). As the RM used by Toro Icarte, Klassen, Valenzano and McIlraith (2022) is more “permissive” (in the sense of (Varricchione, Alechina, Dastani and Logan, 2024)) than the synthesised RM, we speculate that the Toro Icarte, Klassen, Valenzano and McIlraith (2022) RM

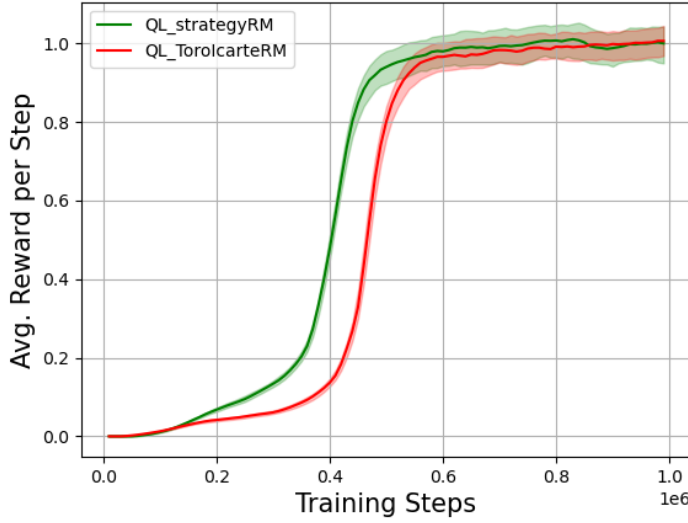
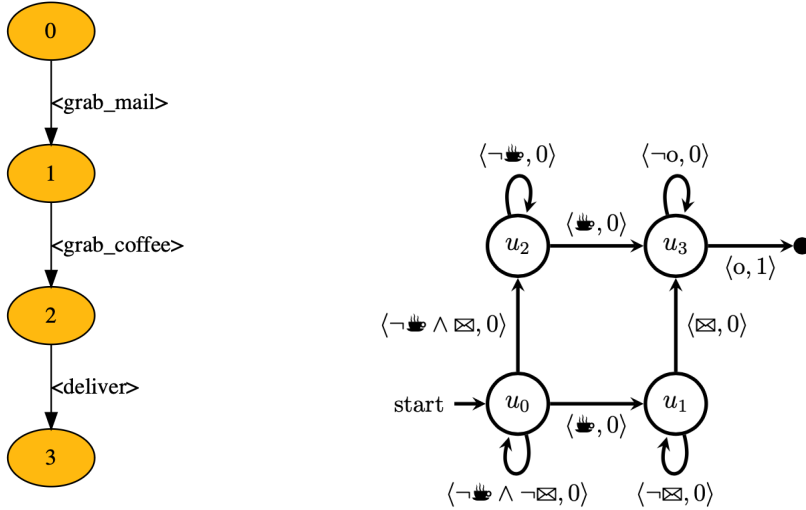


Figure 5.9: Single-agent OFFICEWORLD task (Toro Icarte, Klassen, Valenzano and McIlraith, 2022), agents trained with Q-learning.



(a) Witness for the OFFICEWORLD task; the RM is obtained by labelling the transitions with the corresponding post-conditions.

(b) RM of Toro Icarte, Klassen, Valenzano and McIlraith (2022) for the OFFICEWORLD task.

Figure 5.10: Comparison between the witness used to automatically synthesise our RM and the RM used by Toro Icarte, Klassen, Valenzano and McIlraith (2022) for the OFFICEWORLD task.

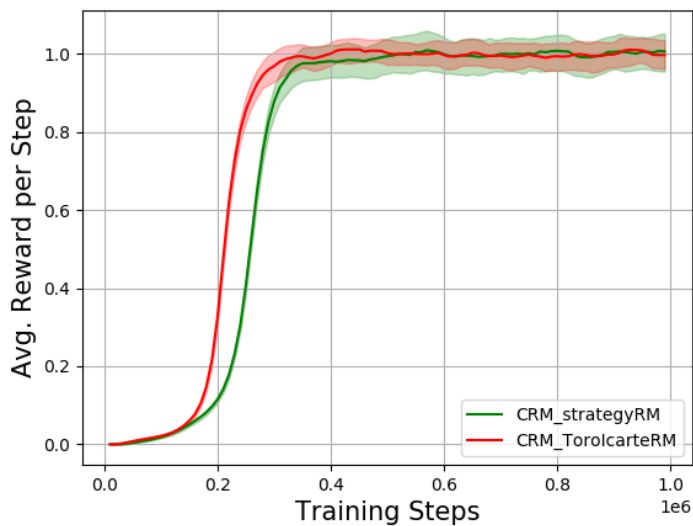


Figure 5.11: Single-agent OFFICEWORLD task (Toro Icarte, Klassen, Valenzano and McIlraith, 2022), agents trained with CRM.

may result in policies achieving better rewards if it is more convenient for the agent to, e.g., first grab coffee and then mail (note how in our RM the agent must first grab mail and then coffee). A more permissive RM can also be beneficial when using algorithms specifically tailored to exploit the structure of reward machines. For example, Figure 5.11 shows the performance of agents trained using the counterfactual RM (CRM) algorithm (Toro Icarte, Klassen, Valenzano and McIlraith, 2022). As it can be seen, the RM of Toro Icarte, Klassen, Valenzano and McIlraith (2022) converges faster than the synthesised RM when using CRM. We believe this is because, at each step, CRM generates 3 experiences when using the synthesised RM and 4 experiences for the Toro Icarte, Klassen, Valenzano and McIlraith (2022) RM.

5.4.4 WATERWORLD

Finally, we consider the WATERWORLD environment, to show how our automatically synthesised reward machines can also be used in a continuous domain. WATERWORLD is a continuous domain where agents (called “*pursuers*”) have to coordinate to capture evaders.⁹ In our case, we have two pursuers and three evaders. Pursuers and evaders are circles that move in a 2D continuous delimited space (see Figure 5.12 for a visualisation of the environment). Pursuers can change their speed and direction, whereas evaders cannot: when the environment is initialized, each evader is initialized with a direction vector and a fixed speed. Each pursuer has a set of evenly spread sensors that allows it to perceive the environment; each sensor reports the relative position (with respect to the pursuer) and the velocity (represented as a two-dimensional vector of real numbers) of the evaders it has perceived. In our experiments, the pursuers have 60 sensors and their range is large enough to perceive all evaders from any position of the space. A pursuer “*captures*” an evader whenever it touches the evader. We consider two different tasks of increasing difficulty requiring different levels of coordination in individual agent tasks and the joint task. The main goal of these experiments is to demonstrate that our approach can be applied in continuous domains.

In the experiments, we used two pursuers A_1 and A_2 , and three evaders F_1 , F_2 , and F_3 , and two tasks of increasing difficulty. In the first (simple) task agent A_1 had to capture agent F_1 , then agent A_2 had to capture evader F_2 , and finally both agents A_1 and A_2 had to capture evader F_3 at the same time. Figure 5.13 shows the witness from which the coalition RM for this task is obtained. In the second (complex) task agents A_1 and A_2 had to respectively capture evaders F_1 and F_3 , then they had to respectively capture evaders F_3 and F_1 , and finally (as in the first task) they both had to capture evader F_2 at the same time.

We compared a pair of pursuer agents trained using our decentralised approach against a “baseline” pair which were trained using a centralised approach without a reward machine but where the rewards obtained by the agents were the same as those given by the coalition RM. We used the WATERWORLD implementation from PettingZoo (Terry et al., 2021). The agents were trained using Proximal Policy Optimization (PPO) (Schulman, Wolski et al., 2017) in both experiments. For the PPO

⁹WATERWORLD was also used by Toro Icarte, Klassen, Valenzano and McIlraith (2022), although in a single-agent setting.

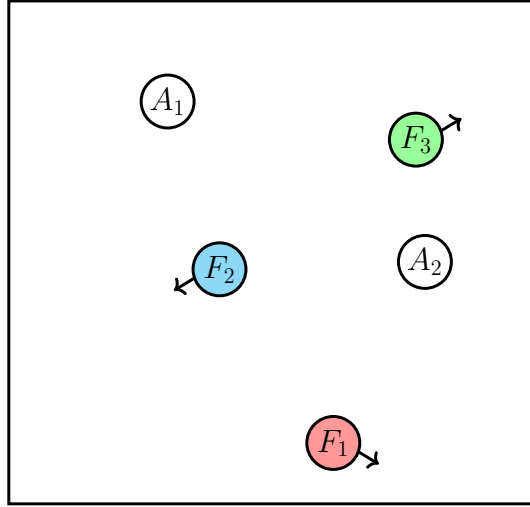


Figure 5.12: WATERWORLD rendition (positions of pursuers and evaders are set randomly at the start of each episode).

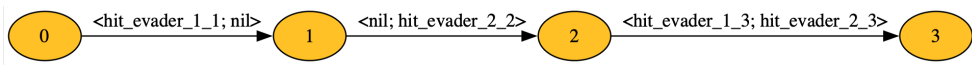


Figure 5.13: Witness for WATERWORLD's "simple" task. In the transition labels, the first number is the pursuer's and the second the evader's.

implementation, we used the RLlib library (Liang et al., 2018) with the following set of parameters: the discount factor $\gamma = 0.99$, the generalized advantage estimation parameter $\lambda = 0.99$, the PPO clip parameter was set to 0.3 in the simple task and 0.1 in the complex task, the coefficient of the value function loss was set to 0.7, and the learning rate is initialized at $\alpha = 10^{-5}$ and linearly decayed to $\alpha = 10^{-9}$ in the first 3,000,000 training steps (after which it remained constant). These values were selected using hyperparameter tuning. RLlib counts a timestep in multi-agent environments every time an agent takes an action. Therefore, in order to not give an advantage to the decentralised approach (where there is only one pursuer agent in the environment which moves at each timestep) over the baseline (where each agent moves at consecutive timesteps), the length of training and of the episodes when training agents individually is halved compared to when they are trained with the centralised baseline. In the decentralised approach, the two agents were each trained for 10,000,000 timesteps; training was divided in episodes which lasted at most 2,000 timesteps in the simple task and 5,000 timesteps in the complex task. For the centralised baseline, training lasted 20,000,000 steps; training was divided in episodes which lasted at most 4,000 timesteps in the simple task and 10,000 timesteps in the complex task. After reaching these limits, in both tasks the environment was reset, the agents were granted no reward, and they had to restart the task. Every 10,000 training steps, we evaluate the policies learned (so far) by the agents by running 10 test episodes. Each of these 10 test episodes is associated to a distinct seed, which is used to initialise the episode during every test. Since the agents act together in the environment during the tests, the test episodes had the same limits on the number of timesteps of the training episodes for the centralised baseline. The WATERWORLD experiment was run on a machine equipped with two 3.0 GHz 16-core AMD 7313 CPUs and a 1024 GB RAM.

The results are shown in Figures 5.14 and 5.15. As it can be seen, the agents trained using the automatically synthesised reward machines obtained better results than the centralised baseline. Indeed, the baseline agents are not able to achieve the task at any point during training. However, in comparison to the COOPERATIVEBUTTONS and RENDEZVOUS domains, the results for the agents trained with a synthesised RM show higher variance in terms of steps needed to achieve the task during the test episodes. In the simple task, there is a difference of approximately 500 steps between the median and the 25th and 75th percentiles, and for the complex task, the difference is approximately 1,000 steps. We hypothesise that this is due to the complexity and randomness of the environment, as WATERWORLD is a continuous domain (both in the state and action spaces) unlike the other domains we have considered.

5.4.5 Summary of Results

The results indicate that, at least for the environments and tasks considered, our automatically synthesised RMs are comparable to RMs given in the literature in terms of sample efficiency. The 25 agent RENDEZVOUS experiment also suggests that our approach is scalable to the numbers of the agents typically considered in the multi-agent reinforcement learning literature, and the WATERWORLD experiment shows that our approach can be applied in continuous domains with very large state

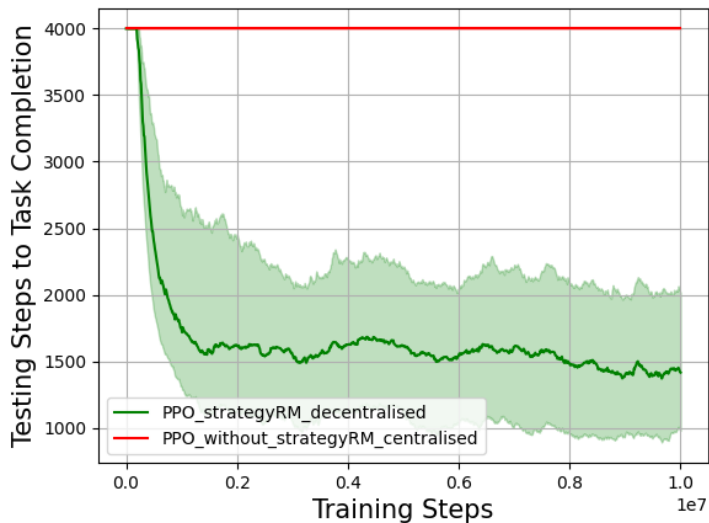


Figure 5.14: Two-agents WATERWORLD “simple” task. Timeout is set to 4,000 timesteps, at which point the task is considered failed.

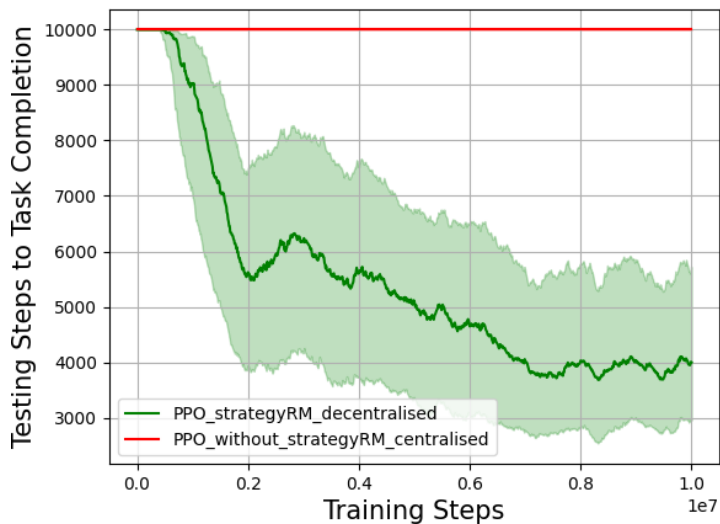


Figure 5.15: Two-agents WATERWORLD “complex” task. Timeout is set to 10,000 timesteps, at which point the task is considered failed.

spaces, where baseline MARL approaches are unable to learn the task. However, the results for OfficeWorld suggest that witness generation techniques which are able to generate more “permissive” strategies (Varricchione, Alechina, Dastani and Logan, 2024) may be beneficial in exploiting RL algorithms tailored for reward machines.

5.5 Conclusions

We have given a procedure to synthesise coalition reward machines for cooperative MARL tasks from specifications given in a fragment of ATL. Then, by decomposing the coalition RM into individual RMs, we can train agents individually, avoiding the issue on non-stationarity as in the approach proposed by Neary et al. (2021). Moreover, for each kind of ATL coalitional formula we consider, we have provided theoretical bounds on the probability that after agents have been trained individually, the coalition will complete the task specified by the ATL formula. In a comparison with previous approaches, we have shown empirically that the performance obtained by using our automatically synthesised RMs is comparable to that obtained by using the RMs of Neary et al. (2021) and Toro Icarte, Klassen, Valenzano and McIlraith (2022). Moreover, our experiments in the WATERWORLD domain suggest that our approach can also scale to more complex, continuous domains.

As for future directions, one would be to investigate whether the use of multiple or “partially-ordered” strategies improves performance in the multi-agent setting. The RMs we synthesise are based on witnesses which essentially correspond to sequential plans, as they represent a single strategy. However, Illanes et al. (2019) showed that, for single agent RL, using partial order plans to construct RMs improves performance. Moreover, we, in (Varricchione, Alechina, Dastani and Logan, 2024), showed theoretically (and empirically in a Minecraft-inspired domain) how an RM obtained from the set of all partial-order plans to achieve a (single-agent) task results in policies that achieve higher rewards compared to RMs generated from a single plan. In the approach of this work, this would translate to having a witness that, instead of representing a single strategy, includes the possible strategies to achieve the task. This can already be done in the current version of MCMAS. Note that a witness representing all strategies to complete the OFFICEWORLD task, would result in an RM that is identical to the one of Toro Icarte, Klassen, Valenzano and McIlraith (2022), as can be seen in Figure 5.16. Thus, this can also lead to faster training when using counterfactual experiences via CRM, as we noticed in the OFFICEWORLD experiment. While this approach can be easily implemented in a single-agent setting, it is not as trivial in a multi-agent one as it would require a further level of coordination between the agents, which should be able to agree on which strategy to follow (for example, by being able to communicate with each other). Another future direction would be to consider non-cooperative RL scenarios. In these cases, ATL could be easily employed to produce a strategy for the coalition of agents we are interested in. However, analogously to what we noted in the previous paragraph, non-cooperative scenarios would also require a further level of coordination between agents, thus making this extension not as trivially as it can seem.

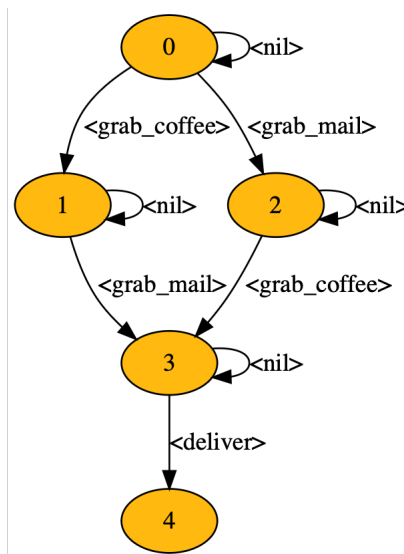


Figure 5.16: Witness showing all strategies to achieve the OFFICEWORLD task.

6 Pushdown Reward Machines for Reinforcement Learning

We now conclude our exploration of reward machines in the dissertation by considering possible extensions of them. As reward machines are, formally speaking, Mealy/Moore machines (depending on how the output function is defined) and thus finite-state automata, they can only recognise and reward behaviours that are “regular”, i.e., that are representable via a regular language. In Chomsky’s hierarchy (Chomsky and Schützenberger, 1959), regular languages are the least expressive formal languages.

There are several tasks that, while natural, are not representable as a regular language. Consider, for example, a Minecraft-like scenario where an agent has to collect as many blocks of wood as possible and then process them in planks in order to build a shelter. With a regular reward machine it is possible to reward the agent for collecting and processing up to n (for a given, fixed, n) blocks of wood. However, this can be sub-optimal (in terms of the time required to achieve the task) if in the environment the agent can gather more than n blocks of wood, as the reward machine would require the agent to collect all the wood in batches of n blocks. The task we have just described can be easily encoded in a deterministic context-free regular language (DCFL), the next class of languages in Chomsky’s hierarchy after regular languages.

DCFLs can be recognised by *deterministic pushdown automata*, an extension of finite-state automata obtained by equipping the latter with a pushdown stack. In this chapter, we present “*pushdown reward machines*” (pdRMs), an extension of reward machines based precisely on the extension of finite state automata to deterministic pushdown automata. We will show how pdRMs can be used to encode tasks in practice, both in discrete and continuous domains, what kind of policies can be learned with them, and how they provide a (theoretical) complexity advantage compared to more expressive extensions of reward machines.

The contents of this chapter are based on (Varricchione, Klassen et al., 2025), which was published in the proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR 2025).

6.1 Introduction

Reward machines (RMs) (Toro Icarte, Klassen, Valenzano and McIlraith, 2018; Toro Icarte, Klassen, Valenzano and McIlraith, 2022) are automata structures that are used to represent (non-Markovian) reward functions for reinforcement learning (RL). Among their merits, they enable RL algorithms to exploit the compositional structure of RMs in learning, resulting in significant sample efficiency gains. By virtue of their correspondence to deterministic finite state automata (DFAs), any reward-worthy behaviour expressible by a regular language, as well as variants of other formal languages such as variants of linear temporal logic (LTL), can be encoded by an RM. This means that a human can write their non-Markovian reward function, or reward-worthy (temporally extended) behaviour in a diversity of programming/formal languages, compile them to an RM, and take advantage of the sample efficiency gains of these RM-tailored learning algorithms (Camacho, Toro Icarte et al., 2019).

A restriction of RMs is that reward-worthy behaviour must be representable in a DFA-like structure—a regular or Type-3 language, according to Chomsky’s hierarchy of languages (Chomsky and Schützenberger, 1959). However, a number of interesting RL problems require the expressiveness of a context-free language. To enhance the expressiveness of RMs, Bester et al. (2024) introduced *counting reward automata* (CRAs) which augment RMs with counters. CRAs have the expressive power of counter machines. As a counter machine with two or more counters has the same expressive power as a Turing machine (Minsky, 1967), CRAs with two or more counters can express behaviours that are representable by recursively enumerable languages, the largest class in the Chomsky hierarchy. Unfortunately, the expressive power of CRAs can come at a significant computational cost for RL. The resulting product MDP and policy incur a blowup depending on the maximum values that the counters can assume. This blowup can severely hinder training by slowing the convergence speed: as the MDP and policy state spaces grow, the time required to explore and learn increases.

In this chapter, following Chomsky’s hierarchy, we propose a more modest enhancement to the expressiveness of RMs by augmenting RMs with a single stack. We call these enhanced RMs *pushdown reward machines* (pdRMs). Our enhancement is based on deterministic pushdown automata (DPDAs), which allow us to encode reward-worthy behaviours that are representable, for example, in LR grammars (Knuth, 1965), or, precisely, deterministic context-free languages. pdRMs can recognize a wide range of practical behaviours, such as modelling recursive calls in programming, collecting and delivering arbitrary numbers of parcels to specific locations, or search and rescue tasks where an agent must return to its starting point by remembering and retracing its (safe) route.

The main contributions of this chapter are as follows:

- We define reward machines based on deterministic pushdown automata. Given their structure, pdRMs can encode tasks representable in deterministic context-free languages;
- We define two variants of pdRM-based policies. In the first, the policy has access to the entire pdRM stack, in the second it can only access the top k symbols (for

a given k) of the stack;

- We provide a procedure to check whether optimal policies with access to the top k stack symbols achieve the same state values as optimal policies with full-stack access;
- We analyse the expressive power of pdRMs and compare it to RMs and CRAs. We also evaluate the space blowup for pdRM- and CRA-based policies. We show that pdRM-policies accessing only the top k symbols of the stack are more compact than pdRM-policies accessing the entire stack and CRA-based policies;
- We propose an approach that exploits counterfactual experiences by generating synthetic experiences based on the states of the pdRM and the stack strings observed during training;
- We use pdRMs to train RL agents in several domains. We compare them to CRAs in a domain from (Bester et al., 2024). Then, we show the practical effects of the space complexity results we establish. We show how counterfactual and hierarchical approaches for pdRMs can be used to increase sample efficiency. Finally, we use pdRMs in a continuous domain, and compare against a deep learning algorithm using recurrent neural networks.

6.2 Pushdown Reward Machines

In this section, we define pushdown Reward Machines (pdRMs). Like RMs, pdRMs are used to express (non-Markovian) reward functions, however their automaton structure is enhanced with a stack, which serves as additional memory. In so doing, they enable the expression of reward-worthy temporally-extended behaviours that correspond to deterministic context-free languages.

Definition 6.1 (Pushdown Reward Machine). A *pushdown reward machine* (pdRM) is a tuple $\mathcal{R} = \langle U, u_0, F, \Sigma, \Gamma, Z, \delta_u, \delta_r \rangle$, where:

- U is the finite set of states;
- $u_0 \in U$ is the initial state;
- $F \subseteq U$ is the set of final states;
- $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ where Σ is the input alphabet;
- $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$ where Γ is the stack alphabet;
- $Z \in \Gamma$ is the initial stack symbol;
- $\delta_u : U \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow (U \cup F) \times \Gamma_\epsilon^*$ is the transition function; and
- $\delta_r : U \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathbb{R}$ is the reward function.

where ϵ is the empty string. Below, we denote by $z \in \Gamma$ individual symbols of the stack alphabet, and by $\zeta \in \Gamma^*$ (possibly empty) stack strings. The transition function δ_u takes as input the current pdRM state u , the current input symbol $\sigma \in \Sigma_\epsilon$ and the topmost symbol of the stack $z \in \Gamma_\epsilon$, and returns a pair (u', ζ') where $u' \in (U \cup F)$ is the next state and $\zeta' \in \Gamma_\epsilon^*$ is a string which replaces z as the topmost symbol(s) of the stack. Note that σ , z and ζ' may be the empty string ϵ . Whenever $\sigma = \epsilon$, the pdRM does not read any symbol from the input, making a so-called “silent” transition. If $z = \epsilon$, no symbol is popped from the stack. Finally, if $\zeta' = \epsilon$, no symbol is pushed to the stack. In what follows, we consider only *deterministic* pdRMs where, for any pdRM state u and stack symbol z , if $\delta_u(u, \epsilon, z)$ is defined, then $\delta_u(u, \sigma, z)$ is not defined for any $\sigma \in \Sigma$. The reward function δ_r takes as input the current pdRM state u , the current input symbol σ and the topmost symbol z of the stack and returns a reward.

At each timestep, a pdRM is in a *configuration* $\langle u, z\zeta \rangle \in U \times \Gamma_\epsilon^*$, where u is the state of the pushdown reward machine and $z\zeta$ is the current string on the stack with z as the topmost symbol on the stack. The pdRM reads the current input symbol $\sigma \in \Sigma_\epsilon$, transitions to a new configuration $\langle u', \zeta'\zeta \rangle$ where $\langle u', \zeta' \rangle = \delta_u(u, \sigma, z)$, and outputs a reward $r = \delta_r(u, \sigma, z)$. We write $\langle u, \zeta \rangle \vdash_\sigma \langle u', \zeta' \rangle$ to denote that the pdRM moves from $\langle u, \zeta \rangle$ to $\langle u', \zeta' \rangle$ upon reading the symbol σ .

We illustrate the expressive power of a pdRM using a simple task expressible in a context-free language, which we call “Maze” task. In the task, the agent has to navigate a (gridlike) maze from a starting location to find a treasure and return to the starting point by following the path it traversed to reach the treasure in the reverse direction. The actions available to the agent are $A = \{u, d, l, r\}$, denoting up, down, left, and right respectively, and the task can be defined using the following (deterministic) context-free grammar:

$$\begin{aligned} S &\rightarrow P x \\ P &\rightarrow u P d \mid d P u \mid l P r \mid r P l \mid t \end{aligned}$$

where t denotes the treasure and x the starting point.

The corresponding pdRM is shown in Figure 6.1. The states in the pdRM are $U = \{u_0, u_1, u_2, u_3\}$, the stack alphabet is $\Gamma = \{u, d, l, r\}$, the input alphabet $\Sigma = 2^{\Gamma \cup \{t, x\}}$ and the labelling L is defined as follows:

- $a \in L(s, a, s')$;
- $t \in L(s, a, s')$ if $s \neq s'$ and s' is the location of the treasure;
- $x \in L(s, a, s')$ if $s \neq s'$ and s' is the starting point.

As with standard reward machines, pdRMs can be used to reward the agent in MDPs. The product of an MDP and a pdRM gives rise to an “MDP-pdRM”, defined as follows.

Definition 6.2. A Markov decision process with a pushdown reward machine (MDP-pdRM) is a tuple $\mathcal{T} = \langle S, \mu, S^F, A, T, \gamma, \Sigma, L, \mathcal{R} \rangle$, where $S, \mu, S^F, A, T, \gamma$ are defined as in an MDP, $L : S \times A \times S \rightarrow \Sigma$ is a labelling function, and $\mathcal{R} = \langle U, u_0, F, \Sigma, \Gamma, Z, \delta_u, \delta_r \rangle$ is a pdRM.

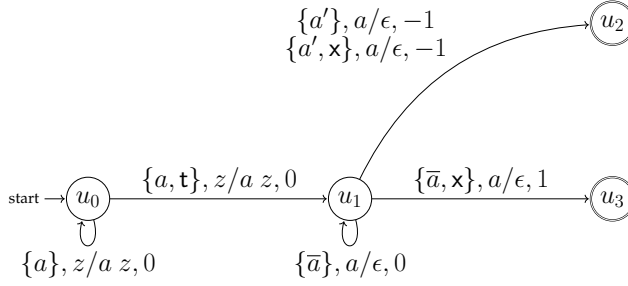


Figure 6.1: Pushdown reward machine for the Maze task. Each transition is labelled with a tuple $\ell, z/\zeta, r$, where ℓ is the input observation, z the top symbol on the stack, ζ the string of symbols pushed onto the stack (with the new top symbol leftmost in ζ), and r is the output reward. The symbol “ z ” indicates an arbitrary symbol in Γ . In the transition from u_1 to u_2 , a' represents any direction except for $\bar{a}$, the opposite direction to a .

Markov decision processes with a pushdown reward machine are analogous to the *Markov decision processes with a reward machine* introduced in (Toro Icarte, Klassen, Valenzano and McIlraith, 2018), but with rewards specified by a pdRM rather than an RM.

An MDP-pdRM induces a *product MDP* $M_{\mathcal{T}} = \langle S_{\mathcal{T}}, \mu_{\mathcal{T}}, S_{\mathcal{T}}^F, A_{\mathcal{T}}, T_{\mathcal{T}}, R_{\mathcal{T}}, \gamma_{\mathcal{T}} \rangle$ where:

- $S_{\mathcal{T}} := S \times (U \cup F) \times \Gamma^*$;
- For arbitrary $s \in S$, we have that $\mu_{\mathcal{T}}(\langle s, u_0, Z \rangle) = \mu(s)$, and that $\mu_{\mathcal{T}}(\langle s, u_0, \zeta \rangle) = 0$ for any other pdRM state $u \in (U \cup F) \setminus \{u_0\}$ or $\zeta \in \Gamma^*$;
- $S_{\mathcal{T}}^F := \{\langle s, u, \zeta \rangle \mid s \in S^F \vee u \in F\}$;
- $A_{\mathcal{T}} := A$;
- $T_{\mathcal{T}}(\langle s', u', \zeta' \rangle \mid \langle s, u, \zeta \rangle, a) := \begin{cases} T(s' \mid s, a) & \text{if } u \in F, u = u' \text{ and } \zeta = \zeta' \\ T(s' \mid s, a) & \text{if } u \notin F \text{ and} \\ & \langle u, \zeta \rangle \vdash_{L(s, a, s')} \langle u', \zeta' \rangle \\ 0 & \text{otherwise} \end{cases}$
- $R_{\mathcal{T}}(\langle s, u, z\zeta \rangle, a, \langle s', u', \zeta' \rangle) := \delta_r(u, L(s, a, s'), z)$;
- $\gamma_{\mathcal{T}} := \gamma$.

Although not specified, the starting state of the pdRM in the product MDP is always u_0 .

We now define the possible policies for MDP-pdRMs. The first type of policy has full access to the current state of the product MDP, and thus to all of the stack's contents.

Definition 6.3 (Policy). A *policy* in an MDP-pdRM is a function $\pi : S \times U \times \Gamma^* \rightarrow \Delta(A)$.

A policy has, at each timestep, access to the current MDP state, pdRM state, and the entire contents of the pdRM stack. Since the stack is potentially unbounded, the state space of policies (hence their size) is potentially unbounded. As such, policies are not well-suited to many RL algorithms, e.g., tabular algorithms, especially in non-episodic tasks (i.e., a task in which the agent must act indefinitely in the environment). Indeed, the large state space can severely hinder training speed, as we will see later in the experimental results.

Motivated by these concerns on the size of policies and on training speed, we define a bounded version of policies which have access only to a portion of the stack. In this way, we bound the size of the policy, making it finite regardless of whether the task is episodic or not. The smaller size of such bounded policies thus allows to train agents faster, given their smaller state spaces.

Definition 6.4 (k -policy). Given a constant $k \geq 0$, a k -policy is a function $\pi : S \times U \times \left(\bigcup_{j \leq k} \Gamma^j\right) \rightarrow \Delta(A)$.

While k -policies can be more suited to learning in RL, we note that their limited observability, similarly to that in a partially observable MDP (POMDP), of the product MDP's state can lead to suboptimal behaviours. In the next section, we will provide a procedure to verify in which cases a k -policy has access to enough information to achieve behaviours with the same state values as optimal policies.

6.3 When Are k -Policies Optimal?

As we have just observed, k -policies can achieve suboptimal behaviours compared to policies, since they have limited access to the product MDP's state. For example, imagine a task in which the agent is given a sequence of rooms (which is saved on the pdRM's stack) that it must clean. Depending on the room, the agent needs different supplies, which are stored in a storage room, to clean it. Gathering all the needed supplies before cleaning the rooms is more efficient as the agent will not have to return to the storage room, and for a policy to anticipate which supplies are needed, it needs access to the entire stack. However, having access to the entire stack is not always necessary to achieve an optimal behaviour. Indeed, in the Maze task it is possible to learn an optimal policy given access to only the top symbol on the stack, as the choice of direction when returning to the starting point is determined only by the top symbol. In this section, we characterise when, for each state, the state value for optimal k -policies is the same as the state value for optimal policies.

First, we define the notion of *k -equivalence* for states in the product MDP induced by an MDP-pdRM.

Definition 6.5 (k -equivalence $\sim_k$). For a product MDP obtained from an MDP-pdRM, two states $\langle s, u, \zeta \rangle, \langle s, u, \zeta' \rangle$, are k -equivalent, denoted by $\langle s, u, \zeta \rangle \sim_k \langle s, u, \zeta' \rangle$, if and only if the top k symbols of ζ and ζ' are the same.

k -equivalence is an equivalence relation which partitions the state space of the product MDP into equivalence classes, where the members of each class share the MDP state, pdRM state, and top k symbols of the pdRM stack. In k -policies, the agent's policy is defined over the set of equivalence classes of the $\sim_k$ relation. We can therefore check whether optimal k -policies have the same value as optimal policies.

The size of the product MDP state space is potentially countably infinite since the states of the product contain the stack, and its size is unbounded. However the rewards are bounded for all states and actions, and depend only on the the pdRM state, the current input symbol and the topmost symbol on the stack. As a result, the potentially unbounded growth in the size of the stack does not affect the immediate rewards. The Bellman operator therefore remains a contraction mapping and value iteration converges to the optimal value function as in the case of finite state spaces (see, e.g., (Feinberg, 2011)).

Proposition 6.1. *If after performing value iteration on the product MDP, all $\sim_k$ -equivalent states have the same value and the same sets of optimal actions, then optimal k -policies achieve the same state values as optimal policies.*

Proof. If the condition in the proposition holds, there exists an optimal policy where action selection depends only on the top k symbols on the stack. Hence, there exists a k -policy that results in the same state values as an optimal policy. $\square$

In practice, it is not possible to check whether the condition in Proposition 6.1 holds for an infinite MDP. However, by bounding the maximal stack size to, e.g., the maximal length of a stack string given the episode length, we can check whether the condition holds for the resulting finite MDP.

6.4 Comparison With CRAs

In this section, we analyse the relative expressive power of pdRMs and counting reward automata (CRA) (Bester et al., 2024) and evaluate the space blowup for pdRM- and CRA-based policies. CRAs are based on counter machines (CM), i.e., finite automata augmented with k counters (for given k) that may be incremented, decremented and tested for zero.

First, we consider the expressive power of pdRMs and CRAs. As both pdRMs and CRAs are automata, the reward functions that they can encode correspond to the languages they accept. Deterministic pushdown automata are strictly more expressive than counter machines with one counter (Fischer, 1966). On the other hand, counter machines with 2 or more counters can simulate a Turing machine (Minsky, 1967), and as such they are strictly more expressive than pdRMs with one stack.

Next, we consider the space complexity of pdRMs and CRAs, where by “space complexity” we mean the bounds on the sizes of policies, i.e., the size of the table representing the policies that are learnt with pdRMs and CRAs. As pdRMs and CRAs only augment the state space of the underlying MDP, we analyse the blowup they cause in the state space to determine the blowup in the size of the policies. To

compare space complexity, we focus on tasks representable in deterministic context-free languages and consider only episodic tasks of length n , as, in the case of non-episodic tasks (i.e., tasks where the agent acts in the environment indefinitely), both stacks and counters can assume infinitely many values and policies learnt with both pdRMs and CRAs may be infinite.

Policies As policies have access to the entire stack, they incur a blowup depending on the number of possible strings that can appear on the stack (i.e., the cardinality of the *stack language*). Let m be the maximum number of symbols that can be added to the stack at any transition of the pdRM. We define an ϵ -sequence as a sequence of ϵ -transitions in the pdRM which are taken before the pdRM must read a symbol to advance. Let e be the maximum number of ϵ -transitions pushing symbols to the stack in a single ϵ -sequence of the pdRM. At each step, the pdRM can add at most $m(e + 1)$ symbols to the stack, and, for input words of length n , the maximum length of the stack string is bounded by $nm(e + 1)$. Hence, the cardinality of the stack language is $|\Gamma|^0 + |\Gamma|^1 + \dots + |\Gamma|^{nm(e+1)}$. If $|\Gamma| > 1$, then the number of stack strings is exactly $\frac{|\Gamma|^{nm(e+1)+1} - 1}{|\Gamma| - 1} \in O(|\Gamma|^{nm(e+1)})$; for $|\Gamma| = 1$ we have that the number of stack strings is $nm(e + 1) + 1 \in O(nm(e + 1))$.

Thus, we have the following:

Theorem 6.2. *If $|\Gamma| \geq 2$, policies incur a blowup in $O(|\Gamma|^{nm(e+1)})$. If $|\Gamma| = 1$, policies incur a blowup in $O(nm(e + 1))$.*

k -policies Unlike for policies, for k -policies we just have to evaluate the number of strings of length at most k made of symbols from the stack alphabet, regardless of the task encoded by the pdRM. Clearly, the number of stack strings of length up to k is $|\Gamma|^0 + |\Gamma|^1 + \dots + |\Gamma|^k$. Then, similarly to policies, if $|\Gamma| > 1$, the number of stack strings is exactly $\frac{|\Gamma|^{k+1} - 1}{|\Gamma| - 1} \in O(|\Gamma|^k)$, and for $|\Gamma| = 1$ the number of stack strings is $k + 1 \in O(k)$. Note that for $k = 1$, this implies that the blowup is linear in $|\Gamma|$. Hence, we get the following.

Theorem 6.3. *If $|\Gamma| \geq 2$, k -policies incur a blowup in $O(|\Gamma|^k)$. If $|\Gamma| = 1$, k -policies incur a blowup in $O(k)$.*

Counting Reward Automata policies A policy trained with access to a CRA (CRA *policy*) has access to the values stored in all counters at each timestep. To evaluate the blowup in the size of the CRA policy, we analyse the number of possible combinations of counter values, which we call “counter configurations”.

First of all, for any DPDA recognising some language L , it is possible to define a CM that recognises the same language by simulating the DPDA. The set of states and the state-transition function of the CM are the same as those of the DPDA. In order to simulate the stack, we need to encode each possible stack string in a counter configuration. As we noted in the discussion of MDP-pdRM policies above, the cardinality of the stack language, and thus the number of counter configurations needed, is in $O(|\Gamma|^{nm(e+1)})$ (where m and e are as in the argument for Theorem 6.2). This implies the following.

Theorem 6.4. *For any task representable as a DCFL, there always exists a CRA which encodes it and incurs a blowup of $O(|\Gamma|^{nm(e+1)})$ in the size of the CRA policy.*

The above result gives an upper bound on the number of possible counter combinations. We note that there is no general tight lower bound on the number of counter configurations for counter machines with k counters (k -CM) to recognise arbitrary DCFLs. Below, we give an example of a DCFL for which any k -CM recognising it requires at least exponentially many (in the input's length) counter configurations.

Let $L_p = \{\sigma x \sigma^R \mid \sigma \in \{0, 1\}^*\}$ be a “marked palindrome” language, where σ^R is the string σ but reversed. We show that, for any k -CM that recognises L_p , the number of counter configurations is at least exponential in the length of the input word. We recall an argument given in the proof of Theorem 1.3 in (Fischer et al., 1968). For any k -CM to recognise L_p , it must be the case that after reading the first half of the word (i.e., before the “mark” x), the k -CM must be in different counter configurations after reading two distinct strings σ_1, σ_2 of length m . This can happen if and only if the number of possible counter configurations (after reading words of length m) is in $\Omega(2^m)$, as we need to encode each string in a tuple of natural numbers. However, in the (worst) case that the string does not contain the mark x , the CM must encode the entire string of length n in a combination of counter values. Therefore, in this second case the number of counter configurations is in $\Omega(2^n)$. Thus, we obtain the following result on the size of CRA policies.

Theorem 6.5. *There is a task representable by a DCFL for which any CRA that encodes it incurs a blowup of $\Omega(2^n)$ in the size of the CRA policy.*

In summary, pdRMs can incur an exponential, with respect to the episode's length, blowup in the size of policies. On the other hand, for k -policies the blowup is polynomial (and linear when $k = 1$) with respect to the cardinality of the stack alphabet. For CRAs, we can always obtain an upper bound for any task expressible as a DCFL by simulating a pdRM for the task (Theorem 6.4).

We have seen a case in which a CRA policy incurs a blowup which is at least exponential in the length of the episode (Theorem 6.5). However, in principle there is no tight lower bound on the blowup for policies based on CRAs, and pdRMs. For example, it is easy to see that the number of strings in the stack language of a pdRM which recognises the language $\{0^n 1^n \mid n \in \mathbb{N}\}$ is linear in the length of the input word. Similarly, one can prove the same bound for CRAs. Therefore, in this case, both policies and CRA policies would incur only a linear (in the episode's length) blowup in the policies' size. Because of this, whether CRAs or pdRMs should be used to encode a given task should be decided on a case-by-case basis. However, if k -policies (for a given k) are sufficient to learn optimal behaviours with respect to the task (see Section 6.3), then k -policies should be preferred to policies if the policy size is to be minimised.

6.5 Exploiting Counterfactual Experiences

RMs were designed, primarily, with two objectives in mind: to provide a normal-form representation for temporally extended reward functions specified natively in RMs

or translated from various languages, and to improve sample efficiency of RL via exploitation of the structure of that normal-form representation. The latter was achieved through the development of off-policy RL algorithms that performed counterfactual reasoning over the automata structure to consider experiences in different automata states. This approach was born out in algorithms such as QRM and CRM which operated in tabular and deep learning settings (Toro Icarte, Klassen, Valenzano and McIlraith, 2018; Toro Icarte, Klassen, Valenzano and McIlraith, 2022), with identical behaviour in tabular domains. At each timestep, these algorithms augment training with the *real* experience with a set of additional synthetic *counterfactual* experiences that correspond to the same (s, a, s') transition experienced in a counterfactual (different) state of the RM. A modified version of this algorithm and approach was proposed by Bester et al. (2024) for use with CRAs.

In this section, we introduce an extension of CRM, which we dub *CpRM*, that can be used with pdRMs. To exploit pdRMs, in addition to considering a counterfactual state of the pdRM, CpRM also changes the string on the stack to produce the counterfactual experiences. However, as we noted in Section 6.4, the number of stack strings can be exponential in the length of the episode. In practice, this can make CpRM extremely slow given the number of counterfactual updates it would make at each timestep. Moreover, observe that in some cases it is even possible that some stack strings are never observed. Because of this, we designed CpRM so that it uses only the set of observed stack strings to produce the counterfactual experiences.

A version of CpRM for policies is shown in Alg. 4 (the approach can be adapted to k -policies by modifying action sampling and policy updates; see Appendix D in Section 6.8). Compared to CRM, the main changes in CpRM lie in lines 2, 6, and 10. In line 2, we initialise the set of observed stack strings $\mathcal{O}$; in line 6 we add the current stack string ζ to $\mathcal{O}$ (obviously, if $\zeta \in \mathcal{O}$ already then this operation changes nothing); finally, in line 10 we cycle through the observed stack strings, and use these to generate the counterfactual experiences. The rest of the algorithm is effectively identical to the original CRM, making it usable with any off-policy RL algorithm. In Section 6.6 we provide results for agents trained with Q-learning augmented with CpRM.

6.6 Experimental Evaluation

In this section we provide empirical results we have obtained from various experiments in five domains.¹ One of the five domains is taken from (Bester et al., 2024) to allow comparison of pdRMs with CRAs. All domains were implemented using the Gymnasium framework (Towers et al., 2024). For discrete domains, we have trained agents using Q-learning (Watkins and Dayan, 1992) bh, both in its vanilla form and with counterfactual updates using CpRM. For the continuous domain we have trained agents with Proximal Policy Optimization (PPO) (Schulman, Wolski et al., 2017) and its recurrent variant, using, respectively, the STABLE-BASELINES3 (Raffin et al., 2021) and SB3 CONTRIB implementations.

¹The code is available at <https://github.com/giovannivarr/pushdown-reward-machines>.

Algorithm 4: Q-learning with CpRM (policy)

Input: MDP-pdRM $\mathcal{T} = \langle S, \mu, S^F, A, T, \gamma, \Sigma, L, \mathcal{R} \rangle$, pdRM $\mathcal{R} = \langle U, u_0, F, \Sigma, \Gamma, Z, \delta_u, \delta_r \rangle$, number of training episodes num_episodes

```

1 For all  $s \in S, u \in U, \zeta \in \Gamma^*, a \in A$  initialise  $Q(\langle s, u, \zeta \rangle, a)$  arbitrarily;
2 Initialize  $\mathcal{O} \leftarrow \emptyset$ ;
3 for  $\ell \leftarrow 0$  to num_episodes do
4    $\langle s, u_0, Z \rangle \leftarrow \mu$ ;
5   while  $s \notin S^F$  and  $u \notin F$  do
6      $\mathcal{O} \leftarrow \mathcal{O} \cup \{\zeta\}$ ;
7     Choose action  $a$  from  $\langle s, u, \zeta \rangle$  using policy derived from  $Q$ ;
8     Take action  $a$  and observe the next state  $s'$ ;
9     for pdRM state  $\bar{u} \in U$  do
10      for observed stack string  $\bar{z}\bar{\zeta} \in \mathcal{O}$  do
11         $\langle \bar{u}', \bar{\zeta}' \rangle \leftarrow \delta_u(\bar{u}, L(s, a, s'), \bar{z})$ ;
12         $\bar{r} \leftarrow \delta_r(\bar{u}, L(s, a, s'), \bar{z})$ ;
13        if  $s' \in S^F$  or  $\bar{u}' \in F$  then
14           $Q(\langle s, \bar{u}, \bar{z}\bar{\zeta} \rangle, a) \xleftarrow{\alpha} \bar{\delta}_r$ ;
15        end
16        else
17           $Q(\langle s, \bar{u}, \bar{z}\bar{\zeta} \rangle, a) \xleftarrow{\alpha} r_c + \gamma \max_{a' \in A} Q(\langle s', \bar{u}', \bar{\zeta}'\bar{\zeta} \rangle, a)$ ;
18        end
19      end
20    end
21    Update  $\langle s, u_0, \zeta \rangle \leftarrow \langle s', u', \zeta' \rangle$ , where  $\langle u', \zeta' \rangle \vdash_{L(s, a, s')} \langle u, \zeta \rangle$ ;
22  end
23 end

```

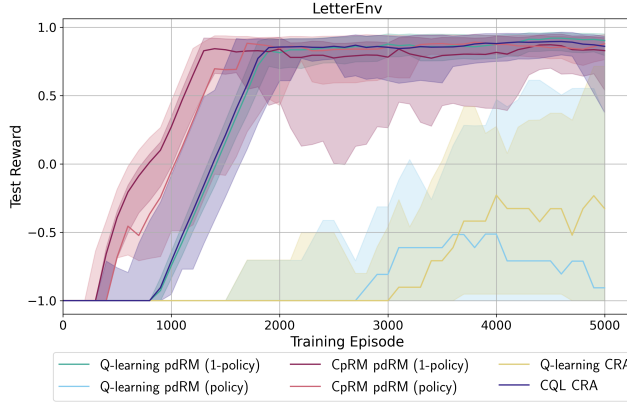


Figure 6.2: LETTERENV results, comparing agents trained with a pdRM and with a CRA. This shows how pdRMs can be used to encode part of the tasks encodable by CRAs.

During experiments, agents were trained and then evaluated by periodically running 10 test episodes after a domain-specific number of training episodes. For all experiments we plot the rewards obtained by the agents in the test episodes, normalised between 1 and -1 . Specifically, we plot the median rewards with lines, and the 25th and 75th percentiles of the rewards as shadowed areas. For details on the experimental setup, specifications of the machines used, and the pdRMs we have used, we refer the reader to Appendix E in Section 6.8.

6.6.1 LETTERENV

The LETTERENV domain is a domain introduced in the original paper on counting reward automata (Bester et al., 2024). The environment is a gridworld where the agent can observe three different events A , B , and C in specific cells. Initially, only the events A and C can be observed. Every time that the agent observes the event A by visiting the corresponding cell, there is a probability of $\frac{1}{2}$ that the cell becomes labelled with the event B from that timestep onwards. The task consists in repeatedly observing the event A until the agent observes the event B , and then observing the event C for the same number of times that the event A was observed. We include this domain to show that pdRMs can be used to obtain comparable results to CRAs when encoding tasks representable in deterministic context-free languages (DCFLs).

For this experiment, we have used the implementation and experimental setup from Bester et al.’s Github repository as of May 12th 2025.² Additional documentation is also available online.³ We trained six agents. Two were trained with the CRA from the repository of Bester et al., one using Q-learning and the other using CQL, a variant

²<https://github.com/TristanBester/counting-reward-machines>

³<https://crm-74a68705.mintlify.app>

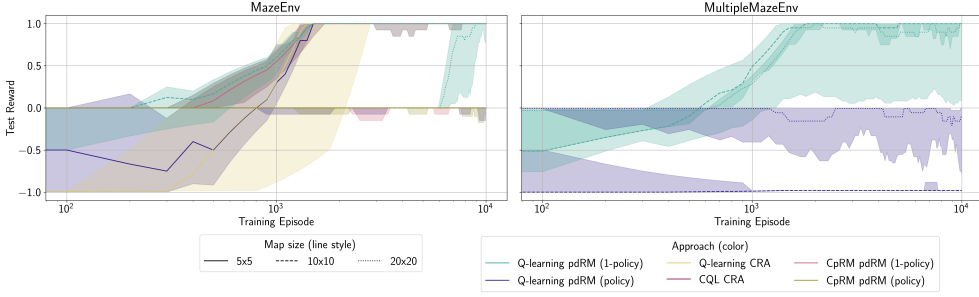


Figure 6.3: 1-TREASUREMAZE (left) and MULTIPLETREASUREMAZE (right) results. We provide individual plots for each maze in Appendix E of Section 6.8. In both plots, each maze is identified by a line style, and each agent by a colour. The 1-TREASUREMAZE plot shows how the 1-pdRM agent could achieve the task on all mazes, whereas the pdRM and the Q-learning CRA agents only on the smallest maze. The -CpRM agents only managed to achieve the task in the smallest maze; in the other cases training timed out due to the time required to perform the policy updates. The CQL-CRA agent never managed to achieve the task. For the MULTIPLETREASUREMAZE experiment, we include only results from the 1-pdRM and pdRM agents trained with vanilla Q-learning. As can be seen, on mazes the 1-pdRM agent learnt to achieve the task, whereas the pdRM agent did not. We believe this is due to the smaller size of the 1-policy.

of the original QRM algorithm (Toro Icarte, Klassen, Valenzano and McIlraith, 2022) that uses counterfactual experiences to improve sample efficiency, adapted by Bester et al. (2024) for CRAs. We refer to the two CRA-based agents as the *Q-CRA agent* and *CQL-CRA agent* respectively. For the pdRM-based agents, we trained a 1-policy and a policy agent with vanilla Q-learning, and a 1-policy and a policy agent with CpRM Q-learning. We refer to these agents as the *1-pdRM agent*, *pdRM agent*, *1-CpRM agent*, and *CpRM agent* respectively, and write *-pdRM* and *-CpRM agents* to refer to all agents trained with the respective variant of Q-learning.

The results of the experiment are shown in Fig. 6.2. As can be seen, the agents that achieve the best performance are the -CpRM agents, the 1-pdRM agent, and the CQL-CRA agent. In contrast, the Q-CRA and the pdRM agents manage to increase their accrued rewards only towards the end of the training episodes; however, they both cannot match the performance of the other agents (although, given enough time, they will eventually reach the same performance). For the agents trained with the pdRM, the results suggest that the smaller state space of the 1-pdRM agent allows it to more easily learn to achieve the task compared to the pdRM agent; moreover, the smaller state space allow the 1-pdRM agent to converge as fast as the CQL-CRA agent, which is trained with counterfactual experiences. Note how the 1-CpRM and the CpRM agents converged faster than the rest, and how the latter agent achieved the task as opposed to the pdRM agent which did not. This shows that CpRM can increase sample efficiency. The results suggest that pdRMs can be used in place of CRAs when tasks can be encoded in DCFLs.

6.6.2 1-TREASUREMAZE

In the 1-TREASUREMAZE environment, the goal of the agent is to navigate a maze to find a treasure, retrieve it, and then return to the initial location by following the reversed path it took to find the treasure. For a detailed explanation of the environment, we refer the reader to the example in Section 6.2. The aim of this experiment is to show there are cases where pdRMs are more sample efficient than CRAs.

For this experiment, we trained the same agents as in the LETTERENV experiment. The CRA encodes the task by encoding the path the agent is following to reach the treasure, similarly to the CM for the marked palindrome language in Section 6.4. At each step, the counters of the CRA are updated so that their configuration correctly encodes the path taken so far. As there are four possible directions identifying each step in the path, the number encoding a path is the translation in base 10 of a number in base 4. We define 0, 1, 2, and 3 to be respectively the directions u, d, l, and r. Thus, at step i , we add to the current encoding of the path the value 4^i times the direction's value. In order to add this value, note how the CRA must repeatedly add 4's, as it can only add constant values in each of its transitions.

Three different mazes were used: a 5×5 maze, a 10×10 maze, and a 20×20 . In this experiment, we end episodes when either the maximum number of timesteps has elapsed, or a limit on wall clock time has been reached. In the left plot in Fig. 6.3, we show the results obtained on all mazes for the 1-TREASUREMAZE experiment. We identify each agent with a colour, and each maze with a line style. In Appendix E of Section 6.8, we provide further details on the maximum timesteps and wall clock limit per episode, and three further plots, each containing the results of the agents on each of the three mazes.

As can be seen (left plot in Fig. 6.3), all the -pdRM agents managed to achieve the task in the 5×5 maze. However, only the 1-pdRM agent managed to learn to also achieve the task in the 10×10 and 20×20 mazes. We believe this is due to the fact that the state space for the 1-pdRM agent is much smaller than that for the pdRM agent. The -CpRM agents could not learn to achieve the task as the training episodes ended prematurely due to the wall clock time limit. This is because of the number of counterfactual updates, which in this scenario is large given the size of the stack language. Note that it can be shown using the approach in Section 6.3 that optimal 1-policies learnt by the 1-pdRM agent have the same state values as optimal pdRM and -CpRM policies.

The Q-CRA agent learned to achieve the task in the 5×5 maze, but not in the larger mazes. We believe the agent could not learn to achieve the task in the larger mazes because the counter values incur an exponential blowup with respect to the number of elapsed timesteps, resulting in an exponential number of operations per timestep and the episodes to end prematurely due to the wall clock time limit. Finally, the CQL-CRA agent never learnt to achieve the task. In addition to having the same issue with respect to the exponential blowup in the number of CRA operations that we observed in the Q-CRA agent, the CQL-CRA agent's training was further slowed down by the counterfactual policy updates. There are exponentially many counterfactual policy updates as the number of possible combinations of counter values is exponential in

the maximum number of timesteps (due to an argument similar to the one we have given to establish Theorem 6.5).

In summary, the experiment shows that there are scenarios where pdRMs are more suited to encode tasks and train agents than CRAs. Note that we do not claim this to always be the case, and believe that it should be decided on a case-by-case basis which of the two machines is more appropriate.

6.6.3 MULTIPLETREASUREMAZE

The MULTIPLETREASUREMAZE environment is a variant of the previous environment where the agent has to retrieve multiple treasures. Before retrieving the treasures, the agent has to first find an intermediate “safe” location. After finding a safe location, the agent starts looking for treasures: as soon as it finds one, it must return to the safe location by following the reversed path it took to find such treasure. This is repeated until the agent finds all treasures, after which the agent observes a special event informing it that it has found all treasures. The agent must then return from the safe location to the initial exit location, by following the reversed path it took to reach the safe location. We include this experiment to show how pdRMs can enable learning of more complex tasks.

In this experiment we have trained only the -pdRM agents. We have excluded the -CpRM and CRA-based agents because the wall clock time limit would have stopped the training episodes prematurely as in the 1-TREASUREMAZE environment. Only a 10×10 maze and a 20×20 one were used. Note that, for this experiment, the maximum number of timesteps per episode for each maze is larger than in the 1-TREASUREMAZE experiments.

As can be seen (right plot in Fig. 6.3), the 1-pdRM agent managed to eventually achieve the task in both mazes. Given the longer episodes compared to the 1-TREASUREMAZE domain, the agents were also able to learn to consistently achieve the task in a relatively small number of episodes. This shows that a pdRM can also be used in tasks where more complex operations with the stack are required. On the other hand, the pdRM agent did not manage to learn to complete the task by the end of training in either maze. We believe that this is mainly due to the size of the policy, which seems to be too large for the agent to learn in this more complex task.

The results show that, thanks to the flexible amount of stack information that pdRMs allow in defining the agent’s policy, pdRMs allow us to train agents in more complex tasks.

6.6.4 DELIVERWORLD

In DELIVERWORLD, the agent that is supposed to deliver packages to locations. Each location is assigned a “type” (e.g., shopping mall, clothing store, etc.), and there can be multiple locations of the same type. At the start of each episode, the agent observes an event which determines the sequence of delivery location types it needs to visit to achieve the task. This sequence is chosen randomly from a set. The aim of this experiment is to illustrate that there are scenarios where it is beneficial for the agent to have access to more than the top symbol on the stack. In this experiment, we show

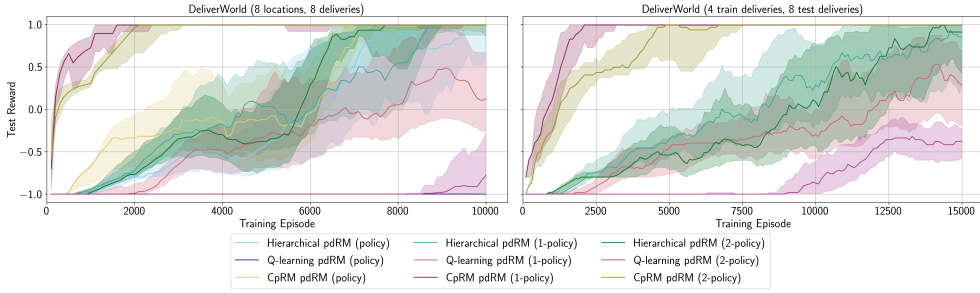


Figure 6.4: DELIVERWORLD results. Left plot: agents performed 8 deliveries during training and testing (DELIVERWORLD-8). Right plot: agents performed 4 deliveries during training episodes and 8 during testing episodes (DELIVERWORLD-4-8). For DELIVERWORLD-8, the only agents that consistently achieve the task at the end of training are the -CpRM agents, the Q-learning 1-pdRM agent and the hierarchical 1-pdRM and 2-pdRM agents. The hierarchical pdRM agent manages to achieve the task but not as consistently. The other Q-learning agents show the worse performance out of all agents, with the pdRM agent flatlining at a reward of -1 . On the other hand, for DELIVERWORLD-4-8, only the -CpRM agents and the hierarchical agents eventually learn to consistently achieve the task. This shows how pdRMs can help in obtaining agents that complete longer tasks than the ones they were trained in.

how CpRM and a hierarchical approach can improve sample efficiency. Moreover, in a variant of the experiment, we show how pdRMs allow training of agents that can perform a larger number of subtasks in testing than in training episodes.

In this experiment we have compared three sets of agents, all trained with access to the same pdRM: one was trained with vanilla Q-learning, one with CpRM Q-learning, and the last with a hierarchical approach.

For the hierarchical approach, we have adapted the hierarchical algorithm proposed in (Toro Icarte, Klassen, Valenzano and McIlraith, 2022). In our pdRM-based version, the meta-policy has access to the current MDP state, pdRM state and the stack of the pdRM. The options' policies have access to the current MDP state, pdRM state, and only the topmost symbol on the pdRM stack.

We have used a 20×20 grid, with two possible setups for the number of deliveries during training and the number of deliveries during testing. In the first, which we call DELIVERWORLD-8, the agents perform 8 deliveries during training and testing episodes. In the second, which we dub DELIVERWORLD-4-8, the agents perform 4 deliveries during training episodes and 8 during testing episodes. In DELIVERWORLD-8, we trained three agents per approach: a 1-policy agent, a 2-policy agent (denoted 2-pdRM agent and 2-CpRM agent for the Q-learning approaches), and a policy agent. In DELIVERWORLD-4-8, we have trained two agents per approach: a 1-policy and a 2-policy agent.

Figure 6.4 shows the plots for both experiments. In both plots, we can clearly see that the most sample efficient agents are the -CpRM ones, followed by the hierarchical ones, and with the vanilla Q-learning ones coming last.

In both scenarios, the -CpRM agents clearly outperform all other agents in speed of convergence. Notice how CpRM and the hierarchical approach improve the performance compared to the pdRM agent in the DELIVERWORLD-8 task: when trained with vanilla Q-learning, the pdRM agent never learns to achieve the task; however the CpRM and hierarchical agent both do. This shows how the hierarchical approach and, when the stack language is not too large, CpRM can greatly increase sample efficiency.

In the hierarchical approach, we can see that in the plot of DELIVERWORLD-8 (left in Fig. 6.4) the 2-pdRM converges faster. We believe that this is due to the fact that, by having access to the top two symbols, the agent learns to correctly plan which delivery location type to visit first. On the other hand, in the plot of DELIVERWORLD-4-8 (right in Fig. 6.4), the 1-pdRM and the 2-pdRM seem to converge at the same pace. For this setup, we think that the fact that the agent needs to perform fewer deliveries during training penalizes agents with larger policy state spaces. Shorter training episodes (compared to the testing ones) imply that the agents can explore fewer states during training, thus leading to worse performance for the 2-pdRM agent compared to the results it obtained in DELIVERWORLD-8.

Finally, for the Q-learning agents, the 1-pdRM agent has the best performance in both plots. In principle, the other two Q-learning agents should be able to learn a policy that is at least as good as that of the 1-pdRM agent. Note that of the Q-learning agents, the only one which is guaranteed to achieve an optimal policy in the limit is the pdRM agent (see Section 6.3). However, given the state-space complexity of the policies and the limited number of episodes, the 2-pdRM and pdRM agents do not learn to achieve the task.

Interestingly, from the plot of DELIVERWORLD-4-8, we can see that agents trained with pdRMs learn policies that, even though trained in smaller instances, performed adequately in larger testing instances. The -CpRM agents are able to achieve the test task relatively quickly, whereas the hierarchical agents eventually manage to achieve the test task by the end of training. Moreover, although the Q-learning agents do not achieve the test task, their testing rewards increase over time. We conjecture that with a longer training they should also converge to policies that achieve the test task. This suggests that pdRMs can, to a certain degree, help agents to learn policies that can achieve longer tasks than the ones that they are trained in.

6.6.5 WATERWORLD

The last domain is based on the WATERWORLD domain (Karpathy, 2015). The task consists of two steps. In the first, the agent must touch 8 balls. Each of these balls is labelled with a (unique) number from 2 to 9: whenever the agent touches the ball identified with i , the pdRM pushes the parity of i to the stack and the ball disappears. Once all 8 balls are touched, the pdRM moves to a new state and the second phase of the task begins. In this phase, the agent has to touch one of two other balls, numbered 0 and 1 determined by the topmost parity on the stack. When the agent correctly touches the ball, it is moved to a new random location and the topmost symbol of the stack is popped. The task is considered achieved when the stack has been emptied.

The aim of this experiment is to evaluate whether pdRMs can provide an advant-

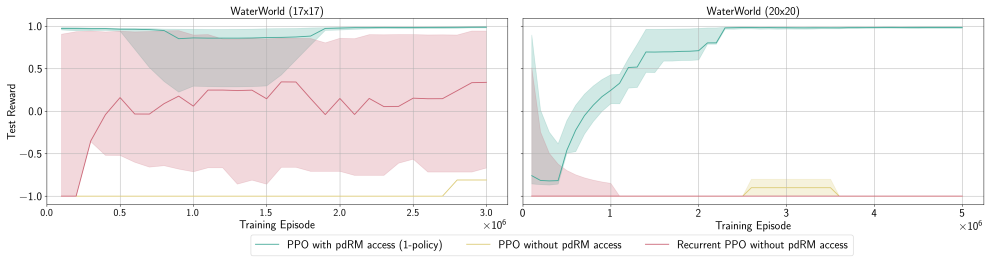


Figure 6.5: WATERWORLD results, 17×17 map (left) and 20×20 map (right). We compare the performance of a 1-pdRM agent trained with PPO against that of an agent trained with PPO and one of an agent trained with recurrent PPO. In the 17×17 map, the 1-pdRM agent is able to achieve the task very quickly. Only the recurrent PPO agent manages to considerably improve its performance, but does not match that of the 1-pdRM agent. Similarly to the results of the 17×17 map, in the 20×20 map the 1-pdRM agent is the only one that consistently achieves the task; significantly outperforming the other agents which do not improve their performance. These two plots show how the pdRM is crucial in training agents to achieve this task.

age in continuous domains when using a deep algorithm. We trained three agents using Proximal Policy Optimization (PPO) (Schulman, Wolski et al., 2017): the first has access to the pdRM that encodes the task (and is trained with a 1-policy), the second does not, and the third also does not have access to the pdRM, but is trained with *recurrent* PPO, i.e., an implementation of PPO where both the actor and critic networks employ *long short-term memory* (LSTM) units (Hochreiter and Schmidhuber, 1997). LSTMs are a type of recurrent neural network, meaning they can capture sequential dependencies in data. Intuitively, using an LSTM should help the agent “remember” important events that have happened during the episode. Note that, as PPO is an on-policy algorithm, CpRM cannot be used. Instead, we used Soft Actor Critic (SAC) in conjunction with CpRM; however, the training of the agents did not terminate, as the server we used to run the WATERWORLD experiments had a time limit on each job. We therefore do not report the results of agents trained with the counterfactual approach for this experiment.

We have trained the agents in two different maps, a 17×17 one and a 20×20 one, and the results are shown in the left and right plot of Fig. 6.5 respectively.

As can be seen, in the 17×17 map (left in Fig. 6.5) the agent trained with the pdRM was able to achieve the task very quickly, and while it showed a small decrease in performance between the 5th and 20th episodes, it recovered and managed to outperform its initial results. The agent trained with recurrent PPO managed to improve its policy, but by the end of training its performance still showed high variance. Finally, the vanilla PPO agent only showed an increase in performance at the end of training, however, its resulting performance was not comparable to that of the 1-pdRM agent or the recurrent PPO agent. In the 20×20 map (right in Fig. 6.5) neither the recurrent PPO nor vanilla PPO agents learnt a policy that achieved the task in the test episodes. However, the agent with access to the pdRM could learn a policy

that consistently achieved the task in the test episodes from the 20th training episode. The results suggest that, in this scenario, using a pDRM is crucial in training and in increasing sample efficiency, performing better than deep algorithms employing recurrent networks.

6.7 Conclusions

In this chapter we have presented *pushdown reward machines*, an extension of reward machines which can encode non-Markovian tasks representable as deterministic context-free languages. Compared to reward machines, pDRMs are thus able to encode a strictly larger set of tasks. We have proposed two policy types for pDRMs, one where the agent has access to the full stack (*policies*), and one where it can access only its top k symbols (k -*policies*). In general, the state values of an optimal k -policy might not be as high as those of a policy. We described a procedure to check whether an MDP and a pDRM are such that the two policy types have the same optimal state values. We have also compared pDRMs to counting reward automata (Bester et al., 2024), another extension of reward machines capable of encoding tasks representable as any recursively enumerable language. We showed that, when an agent trained with a pDRM has access only to the top symbol of the pDRM's stack, the size of a 1-policy can be exponentially smaller (with respect to the episode's maximum length) than the size of a policy an agent trained with a CRA for the same task. Finally, in the experimental evaluation, we have shown how pDRMs can be used in practice. We have seen how in certain scenarios it is more convenient to use pDRMs than CRAs. We have also provided counterfactual and hierarchical algorithms specifically tailored for pDRMs, and saw how they can increase convergence speed. Finally, we have used pDRMs in a continuous domain, and showed how they can outperform state-of-the-art algorithms employing recurrent neural networks.

There are several directions for future work. First, we plan to investigate the mixed performance of CpRM in the 1-TREASUREMAZE and WATERWORLD tasks. It would also be interesting to automatically synthesise pDRMs, as was done with reward machines via, e.g., logic formalisms (Camacho, Toro Icarte et al., 2019; Varricchione, Alechina, Dastani and Logan, 2023), search and learning (Toro Icarte, Waldie et al., 2019; Toro Icarte, Klassen, Valenzano, Castro et al., 2023; Furelos-Blanco et al., 2023; H. Hasanbeig et al., 2024) or planning (Illanes et al., 2019; Varricchione, Alechina, Dastani and Logan, 2024). As deterministic pushdown automata are the underlying structure of pDRMs, we think LR(k) grammars (Knuth, 1965) could be good candidates to synthesise pDRMs, as they can be easily translated into DPDAs (Aho and Ullman, 1973). Finally, as the stack can provide the agent with even further memory, pDRMs can be an interesting alternative approach to RMs (Toro Icarte, Waldie et al., 2019; Toro Icarte, Klassen, Valenzano, Castro et al., 2023) in dealing with partially observable environments.

6.8 Supplementary Material

Appendix A: Example Translation From CRA to pdRM

In this Section of the Appendix, we provide an example of a translation from a 1-counter CRA to a pdRM. Specifically, we consider the CRA given as an example in the work by Bester et al. for the LETTERENV task. The CRA is shown in Fig. 6.6.

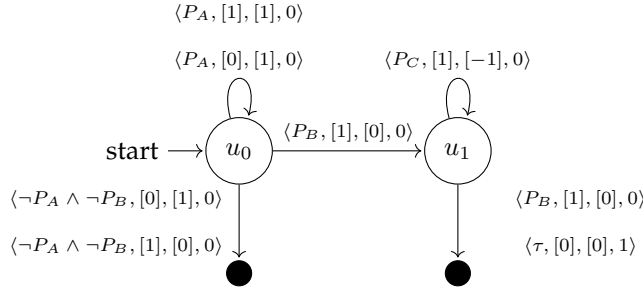


Figure 6.6: CRA for the LETTERENV task. The states drawn as black circles are the terminal states.

An equivalent pdRM can be seen in Fig. 6.7. In it, we use the special symbol $\#$ to denote the end of the stack (we set the initial stack symbol Z to be $\#$). The special symbol $\#$ is used to simulate the zero-test on the counter's value done in the CRA of Fig. 6.6: if the top symbol is $\#$, then the counter's value would be zero; otherwise, if the top symbol is not $\#$, then the counter's value would not be zero.

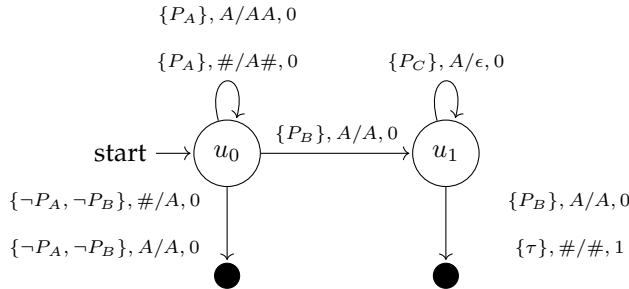


Figure 6.7: pdRM for the LETTERENV task equivalent to the CRA in Fig. 6.6.

Appendix B: General Translation From 1-Counter CRA to pdRM

In Appendix A we have seen an example of a translation from a CRA to a pdRM. In general, it is not possible to translate an arbitrary CRA to a pdRM: as CRA can have multiple counters, they can be strictly more expressive than pdRMs. However,

for CRAs with one counter⁴ there exists always a pdRM which encodes the same non-Markovian reward function. In this Section of the Appendix, we provide a formal construction that, given a 1-counter CRA, produces a pdRM that encodes the same non-Markovian reward function.

Fix a given 1-counter CRA $\mathcal{C} = \langle U, F, \Sigma, \Delta, \delta, \lambda, u_0 \rangle$. From it, we build the corresponding pdRM $\mathcal{R}^{\mathcal{C}} = \langle U^{\mathcal{C}}, u_0^{\mathcal{C}}, F^{\mathcal{C}}, \Sigma^{\mathcal{C}}, \Gamma^{\mathcal{C}}, Z^{\mathcal{C}}, \delta_u^{\mathcal{C}}, \delta_r^{\mathcal{C}} \rangle$ as follows:

- $U^{\mathcal{C}} := U \cup U^{\delta}$, where U^{δ} is a set of “helper” states used to correctly simulate decreases in the counter’s value. Following, when defining the transition function $\delta_u^{\mathcal{C}}$, we will properly define the set U^{δ} ;
- $u_0^{\mathcal{C}} := u_0$;
- $F^{\mathcal{C}} := F$;
- $\Sigma^{\mathcal{C}} := \Sigma \cup \{\epsilon\}$, where ϵ is a symbol not in Σ ;
- $\Gamma^{\mathcal{C}} := \{\#, \square\}$. The $\#$ symbol is used to denote the end of the stack, and similarly to the pdRM from Appendix A will be used to simulate the zero-test of the counter’s value. The $\square$ symbol will instead be used to represent a single unit on the counter: whenever the counter’s value in the CRA is modified by adding/subtracting n , then the stack will be changed by pushing/popping n $\square$ symbols;
- $Z^{\mathcal{C}} := \#$.

We now define the transition and reward functions $\delta_u^{\mathcal{C}}, \delta_r^{\mathcal{C}}$ of the pdRM. In order to define $\delta_u^{\mathcal{C}}$, consider an arbitrary CRA configuration $\langle q, c \rangle$, where q is the CRA state and c the value of the CRA counter. Upon observing the input symbol σ , the CRA transitions to the configuration $\langle q', c' \rangle = \delta(q, \sigma, \mathbb{1}(c))$ and outputs reward $r = \lambda(q, \sigma, \mathbb{1}(c))$. Here, $u, u' \in U$ are states of the CRA, c and c' are the value of the CRA’s counter, $\sigma \in \Sigma$ is the input symbol, and $\mathbb{1}(c)$ is the output of the zero-test function, defined as follows:

$$\mathbb{1}(c) := \begin{cases} 0 & \text{if } c = 0 \\ 1 & \text{otherwise} \end{cases}$$

As we have mentioned before, the zero-test function can be easily simulated in the pdRM by checking whether the top symbol of the stack is $\#$ or not. Hence, to define $\delta_u^{\mathcal{C}}$ and $\delta_r^{\mathcal{C}}$, if $\mathbb{1}(c) = 0$ we require that the top stack symbol in the input of the two function is $\#$, and otherwise that it is $\square$. Following, when defining $\delta_u^{\mathcal{C}}$ and $\delta_r^{\mathcal{C}}$, we let γ be the top symbol as we have just specified.

Note that c' is obtained by adding or subtracting to c some value m . To define $\delta_u^{\mathcal{C}}$ and $\delta_r^{\mathcal{C}}$, we differentiate between the case where $m \geq 0$ and the one where $m < 0$:

⁴It should be noted that here by “CRA” we are referring specifically to the so-called “constant CRAs”, i.e., the CRAs where the output function outputs real values (rewards) and not functions. As shown by Bester et al., the two formalisms are equally expressive.

- $m \geq 0$: in this case, the pdRM stack needs to be modified by pushing m $\square$ symbols. This can be easily done in a pdRM, by pushing the symbols in a single transition. For the transition function, we define $\delta_u^C(q, \sigma, \gamma) = (q', \square^m \gamma)$, where $\square^m$ is a string of m $\square$ symbols. For the reward function, we define $\delta_r^C(q, \sigma, \gamma) = r$.
- $m < 0$: in this case, the pdRM stack needs to be modified by popping m $\square$ symbols. Unlike the previous case, in this one we cannot pop all the m symbols in a single transition, as pdRM allow to only pop one symbol (the topmost on the stack) per transition. We therefore use “helper” states to correctly pop the m symbols. Let $u_{q,\sigma,\gamma}^1, \dots, u_{q,\sigma,\gamma}^{|m|} \in U^\delta$ be the helper states for this transition. Starting from state q , the transition function will first move to the first helper state upon observing σ , and it will then traverse all the helper states until reaching the state q' via ϵ -transitions. While transitioning through the helper states, the transition function will also pop the topmost symbol if it is a $\square$ (if it the $\#$, it will not pop it). In the meanwhile, the reward function will always output 0, until the transition from the last helper state to the state q' when it will output the reward r .

Thus, we define the transition and reward functions as follows:

- δ_u^C : first, transition from state q to state $u_{q,\sigma,\gamma}^1$ without popping the top symbol, i.e., $\delta_u^C(q, \sigma, \gamma) = (u_{q,\sigma,\gamma}^1, \gamma)$. We now differentiate two cases: if $|m| = 1$, then it means that there are no more helper states to traverse and the pdRM should now move to the new state q' . We thus define two transitions from $u_{q,\sigma,\gamma}^1$, depending on whether the top symbol γ is $\square$ or $\#$. If $\gamma = \square$, we should pop the symbol as the value of the CRA’s counter was decreased by 1; if $\gamma = \#$ then we do not pop the symbol as it means that the value of the CRA’s counter was already 0 (recall that the value of a CRA counter is always non-negative). Thus, we let $\delta_u^C(u_{q,\sigma,\gamma}^1, \epsilon, \square) = (q', \epsilon)$ and $\delta_u^C(u_{q,\sigma,\gamma}^1, \epsilon, \#) = (q', \#)$. On the other hand, if $|m| > 1$, then there is at least one other helper state $u_{q,\sigma,\gamma}^2$. Analogously to the previous case, we define two transitions from $u_{q,\sigma,\gamma}^1$ to $u_{q,\sigma,\gamma}^2$, one for $\gamma = \#$ (where $\#$ is not popped) and one for $\gamma = \square$ (where $\square$ is popped). We now repeat the same construction for the rest of the helper states $u_{q,\sigma,\gamma}^3, \dots, u_{q,\sigma,\gamma}^{|m|}$ (if there are any);
- δ_r^C : define $\delta_r^C(q, \sigma, \gamma) = 0$ and $\delta_r^C(u_{q,\sigma,\gamma}^i, \epsilon, \gamma) = 0$ for all $1 \leq i < |m|$ and $\gamma \in \{\#, \square\}$. Then, let $\delta_r^C(u_{q,\sigma,\gamma}^{|m|}, \epsilon, \gamma) = r$ for $\gamma \in \{\#, \square\}$.

From the construction, it should be clear that both the CRA and the pdRM will output the same rewards while reading any input word.

Appendix C: Optimality of Full-Policies

In this section of the Appendix we present an environment and experimental results that show in practice how full-policies can be more optimal than top- k -policies,

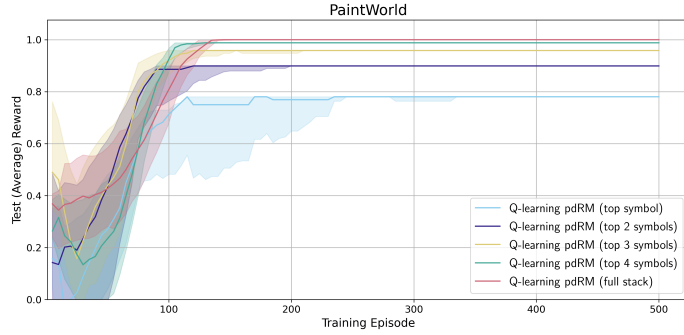


Figure 6.8: PAINTWORLD results. While rewards were normalized between -1 and 1 , here we only show the positive half of the y axis to make the lines more distinguishable.

and how top- k -policies can be more optimal than top- k' -policies, for $k' < k$ (cfr. Section 6.3).

For this experiment, we have devised an environment called “PAINTWORLD”, in which the agent’s goal is to request an adequate amount of soap to clean paint stains. We have encoded the task using a pdRM: whenever an episode starts, the stack is initialized by pushing a string of n paint symbols, where n is randomly sampled and ranges from 1 to 5. The actions the agent can perform consist in requesting i units of soap, for i ranging also from 1 to 5. Whenever the agent asks for i units of soap, it receives a penalty of $\frac{i}{i+1}$. Hence, if there are i paint stains, it is always optimal to ask for precisely i units of soap. The pdRM always outputs 0 as its reward,

To show how agents that have access to more symbols on the stack can achieve better policies (with respect to the achieved reward), we have trained 5 agents, where the i -th agent has access to the top i symbols on the stack. Each agent was trained for a total of 500 episodes, each lasting at most 5 timesteps, and evaluated every 5 training episodes. Agents were tested by running 5 test episodes, where in the i -th test episode the agents had to clean i paint stains. In the plot in Fig. 6.8 we report the average rewards obtained by the agents in each test. As can be seen in the plot, the more symbols the agent can access on the stack, the better the policy it learns. The only agent who achieves the optimal performance is the full-policy agent: indeed, in the test episodes the full-policy agent is the only one which optimally chooses to request i units of soap whenever it needs to clean i stains of paint.

Appendix D: Counterfactual Algorithm for k -Policies

Here we give a variant of the counterfactual algorithm presented in Section 6.5 which can be used to learn a k -policy. In the algorithm that follows, for a given pdRM stack string ζ , we denote with $\zeta_{\upharpoonright k}$ the substring obtained by taking the top k symbols of ζ .

Algorithm 5: Q-learning with CpRM (k -policy)

Input: MDP-pdRM $\mathcal{T} = \langle S, \mu, S^F, A, T, \gamma, \Sigma, L, \mathcal{R} \rangle$, pdRM
 $\mathcal{R} = \langle U, u_0, F, \Sigma, \Gamma, Z, \delta_u, \delta_r \rangle$, number of training episodes
num_episodes

```

1 For all  $s \in S, u \in U, \zeta \in \Gamma^k, a \in A$  initialise  $Q(\langle s, u, \zeta \rangle, a)$  arbitrarily;
2 Initialize  $\mathcal{O} \leftarrow \emptyset$ ;
3 for  $\ell \leftarrow 0$  to num_episodes do
4    $\langle s, u_0, Z \rangle \leftarrow \mu$ ;
5   while  $s \notin S^F$  and  $u \notin F$  do
6      $\mathcal{O} \leftarrow \mathcal{O} \cup \{\zeta\}$ ;
7     Choose action  $a$  from  $\langle s, u, \zeta_{|k} \rangle$  using policy derived from  $Q$ ;
8     Take action  $a$  and observe the next state  $s'$ ;
9     for pdRM state  $\bar{u} \in U$  do
10      for observed stack string  $\bar{z}\bar{\zeta} \in \mathcal{O}$  do
11         $\langle \bar{u}', \bar{\zeta}' \rangle \leftarrow \delta_u(\bar{u}, L(s, a, s'), \bar{z})$ ;
12         $\bar{r} \leftarrow \delta_r(\bar{u}, L(s, a, s'), \bar{z})$ ;
13        if  $s' \in S^F$  or  $\bar{u}' \in F$  then
14           $Q(\langle s, \bar{u}, (\bar{z}\bar{\zeta})_{|k} \rangle, a) \xleftarrow{\alpha} \bar{r}$ ;
15        end
16      else
17         $Q(\langle s, \bar{u}, (\bar{z}\bar{\zeta})_{|k} \rangle, a) \xleftarrow{\alpha} r_c + \gamma \max_{a' \in A} Q(\langle s', \bar{u}', (\bar{\zeta}'\bar{\zeta})_{|k} \rangle, a)$ ;
18      end
19    end
20  end
21  Update  $\langle s, u_0, \zeta \rangle \leftarrow \langle s', u', \zeta' \rangle$ , where  $\langle u', \zeta' \rangle \dashv_{L(s, a, s')} \langle u, \zeta \rangle$ ;
22 end
23 end

```

Appendix E: Experimental Evaluation Details

In this section, we provide, for each experiment, the experimental setup (e.g., environment size, episode length) and the hyperparameters settings.

For each experiment, we report the results obtained by the agents across 10 different training runs. During training, after a set amount of training episodes (which varies across the different experiments), we evaluated the agents by running 10 test episodes. We randomly selected 10 seeds $x_1, \dots, x_{10}$, and for each agent and experiment, the i -th run was seeded with seed x_i .

LETTERENV

We note that for this experiment we have used the set of hyperparameters provided in the most updated version (as of May 7th, 2025) of the counting reward automata's repository (<https://github.com/TristanBester/counting-reward-machines/>). Following are the hyperparameters:

- Learning rate $\alpha = 0.01$;
- Exploration rate $\epsilon = 0.01$;
- Discount factor $\gamma = 0.99$.

As for the environment and experimental setup, we have trained the agents for a total of 5,000 episodes, each lasting at most 300 timesteps. Testing episodes were run every 100 training episodes. The map's size was 3×7 (rows $\times$ columns).

Figure 6.9 contains the pdRM used in this experiment.

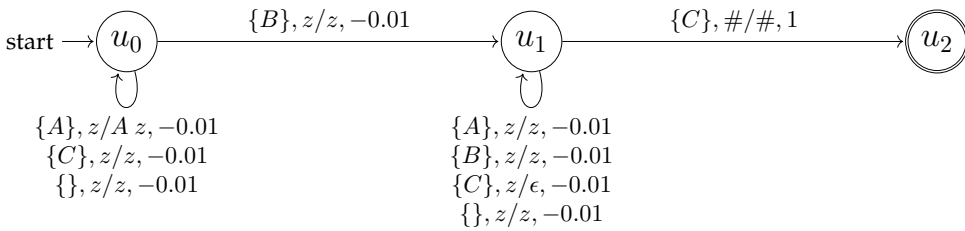


Figure 6.9: Pushdown reward machine for the LETTERENV domain. $\#$ is a special symbol used to denote the start of the stack.

1-TREASUREMAZE

First, we include in Fig. 6.10 a rendition of the smallest maze that we have used in this experiment.

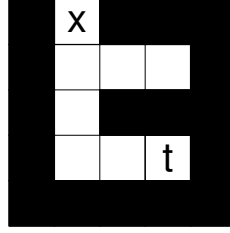


Figure 6.10: Rendition of the 5×5 maze. x is the initial and exit location, t the location of the treasure, and the black cells are walls.

We have used the following set of hyperparameters for this experiment:

- Learning rate $\alpha = 0.5$;
- Exploration rate ϵ initialised at 1, and linearly decayed to 0.01 by multiplying it by 0.995 after every episode;
- Discount factor $\gamma = 0.99$.

Agents were trained for a total of 10,000 episodes and were tested every 100 episodes. The maximum number of timesteps and limit for wall clock times per episode are as follows:

- 5×5 : 15 timesteps, 0.03 seconds per episode;
- 10×10 : 15 timesteps, 0.06 seconds per episode;
- 20×20 : 300 timesteps, 0.1 seconds per episode.

For the time limit, we have measured the wall clock time of the training process by using the Python `time.perf_counter()` function.

Figure 6.11 contains the pdRM used in this experiment.

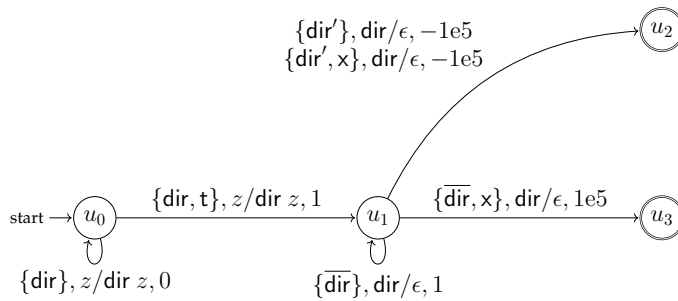


Figure 6.11: Pushdown reward machine for the 1-TREASUREMAZE domain. For an explanation of the labels we refer the reader to Figure 6.1.

As we have used three different maps for this experiment, we include here (Figs. 6.12 to 6.14) also the results on each of these maps.

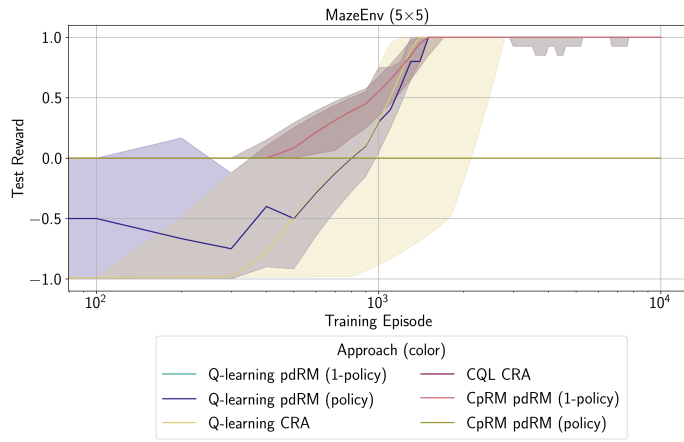


Figure 6.12: 1-TREASUREMAZE results, 5×5 map.

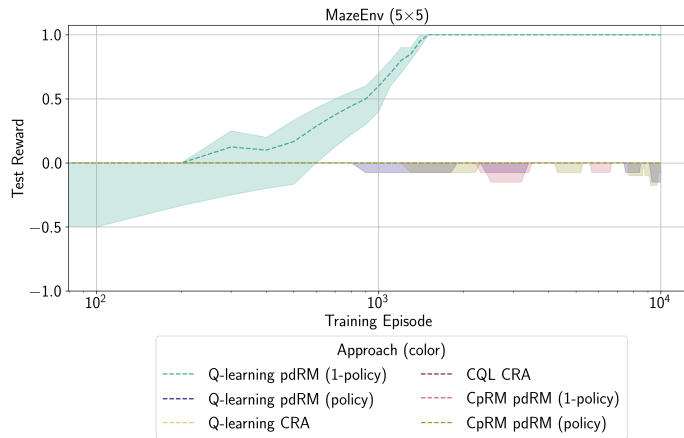


Figure 6.13: 1-TREASUREMAZE results, 10×10 map.

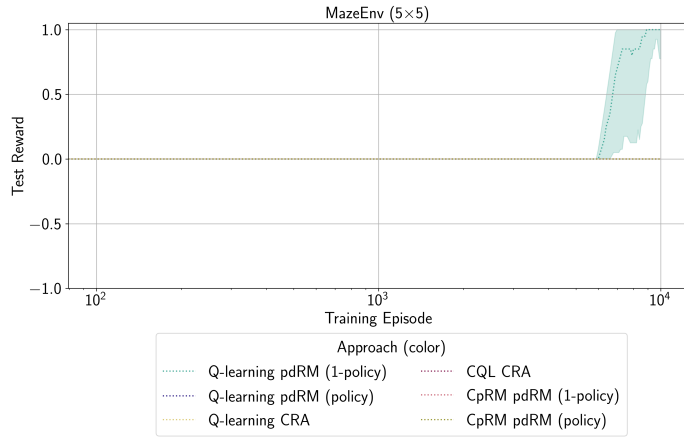


Figure 6.14: 1-TREASUREMAZE results, 20×20 map.

MULTIPLETREASUREMAZE

We have used the following set of hyperparameters for this experiment:

- Learning rate $\alpha = 0.5$;
- Exploration rate ϵ initialised at 1, and linearly decayed to 0.01 by multiplying it by 0.995 after every episode;
- Discount factor $\gamma = 0.9$.

Agents were trained for a total of 10,000 episodes and were tested every 100 episodes. The maximum number of timesteps per episode are as follows:

- 10×10 : 1,000 timesteps;
- 20×20 : 4,000 timesteps.

Figure 6.15 contains the pdRM used in this experiment.

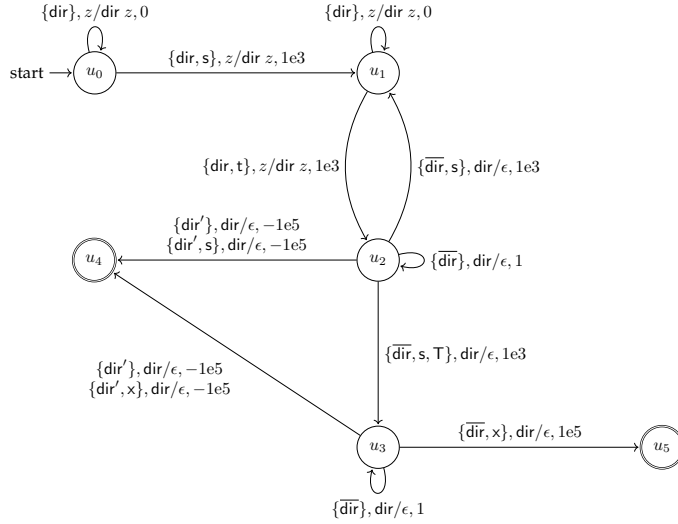


Figure 6.15: Pushdown reward machine for the MULTIPLETREASUREMAZE domain. s is the event observed upon reaching the safe location in the maze, T the one when all treasures have been found.

DELIVERWORLD

For this experiment, we report the hyperparameters to train the policy of the Q-learning agent, and the meta-policy and the options' policies of the hierarchical agent. To train these policies, we used the same set of hyperparameters:

- Learning rate $\alpha = 0.01$;
- Exploration rate ϵ initialised at 1, and linearly decayed to 0.1 by multiplying it by 0.9995 after every episode;
- Discount factor $\gamma = 0.99$.

Agents were trained for a total of 10,000 episodes in the experiment where they performed 8 deliveries both during training and testing episodes, and for 15,000 episodes in the experiment where they performed 4 deliveries during training and 9 deliveries during testing episodes. In both cases, they were tested every 100 episodes. For all experiments, each episode (both during training and testing) lasted at most 2,000 timesteps. At each timestep, the agents received a punishment of $-\frac{1}{2}$ (aside from the pdRM's reward).

Figure 6.16 contains the pdRM used in this experiment.

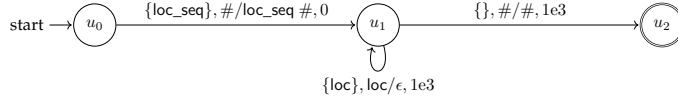


Figure 6.16: Pushdown reward machine for the DELIVERWORLD domain. `loc_seq` is the sequence of location types given in input at the start of each episode. `#` is a special symbol used to denote the start of the stack.

WATERWORLD

First, for the agents that were trained with “vanilla” PPO, we have used the STABLE-BASELINES3⁵ implementation of PPO. Instead, for the agent trained with recurrent PPO, we have used the SB3 CONTRIB⁶ implementation. STABLE-BASELINES3-CONTRIB is the repository extending STABLE-BASELINES3 with community-based implementations of modified or more experimental algorithms.

We report the set of hyperparameters (using the names of their STABLEBASELINES3 variables) that we used for “vanilla” PPO and recurrent PPO. We have used a different set of hyperparameters on the two maps (17×17 and 20×20). Every hyperparameter that is not reported was set to its default value in STABLEBASELINES3 and STABLEBASELINES3-CONTRIB. Finally, we note that the reported hyperparameters were chosen after tuning both algorithms on both maps.

For the two maps and the two agents, we had the following experimental setup. The set of experience samples were collected from 5 copies of the environment. Each training and testing episode lasted at most 10,000 timesteps, and agents were evaluated every 100,000 training timesteps. The learning rate was linearly decayed to 0 (starting from the ones we will report) in all runs.

For the 17×17 map we have trained both agents for a total of 3,000,000 timesteps, and for the 20×20 map for a total of 5,000,000 timesteps. We give the hyperparameters for the agents trained in 17×17 map and in the 20×20 map respectively in Table 6.1 and Table 6.2.

Figure 6.17 contains the pdRM used in this experiment.

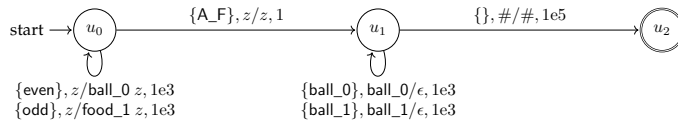


Figure 6.17: Pushdown reward machine for the WATERWORLD domain. `even` and `odd` are the events observed when, respectively, the agent captures a ball (except for the first two) which label has the corresponding parity. `A_F` is the event observed when all balls (except for the first two) have been captured. `#` is a special symbol used to denote the start of the stack.

⁵<https://stable-baselines3.readthedocs.io/en/master/>

⁶https://stable-baselines3.readthedocs.io/en/master/guide/sb3_contrib.html

Hyperparameter	Vanilla PPO	Recurrent PPO
learning_rate	3e−6	3e−9
n_steps	10,000	10,000
batch_size	3,000	2,000
n_epochs	30	20
gamma	0.99	0.95
gae_lambda	0.9	0.95
clip_range	0.15	0.15
clip_range_vf	1,000	1,000
vf_coef	0.5	1.0
target_kl	0.3	0.3
normalize_advantage	True	True

Table 6.1: Hyperparameters for Vanilla PPO and Recurrent PPO in the 17×17 WATER-WORLD map.

Hyperparameter	Vanilla PPO	Recurrent PPO
learning_rate	3e−5	3e−6
n_steps	10,000	10,000
batch_size	3,000	2,000
n_epochs	30	20
gamma	0.99	0.95
gae_lambda	0.9	0.95
clip_range	0.15	0.15
clip_range_vf	1,000	1,000
vf_coef	0.5	1.0
target_kl	0.3	0.3
normalize_advantage	True	True

Table 6.2: Hyperparameters for Vanilla PPO and Recurrent PPO in the 20×20 WATER-WORLD map.



7 Pure-Past Action Masking

We now diverge from the previous three chapters and consider the problem of training RL agents in a *safe* manner, i.e., in such a way that they respect *safety* constraints as required by users. This chapter will thus focus on “*safe reinforcement learning*” (García and Fernández, 2015), a branch of RL where it is assumed that there are safety constraints that the agent must satisfy or, at least, not violate with a high probability. This sub-area is usually concerned with more safety-critical applications, such as autonomous cars, or domains where the cost of equipment is high and the chance of failures must be therefore minimised, such as chemical processes and robotics.

The work on which this chapter is based makes use of a variant of Linear-time Temporal Logic, which is called “*Pure-Past Linear-time Temporal Logic*” (PPLTL) (De Giacomo, Di Stasio et al., 2020). As the name suggests, PPLTL is a variant of LTL where the temporal modalities consider exclusively what has happened so far, and are not concerned with what will happen in the future. Indeed, instead of having the typical LTL modalities, e.g., next X or until U , we have modalities such as “*Yesterday*” Y or “*Since*” S . The main idea of the approach presented in this chapter is to use PPLTL formulas to constrain the actions that the agent can select. For example, if an agent is supposed to take a bus, then it should not be allowed to decide to board the bus unless it has bought a ticket previously. As we will see, by using PPLTL it is possible to enforce safety constraints, such as the one that we have just mentioned, and guarantee that the agent will satisfy them, both during and after training.

The contents of this chapter are based on (Varricchione, Alechina, Dastani, De Giacomo et al., 2024), which was published in the proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024), in the technical track on *Safe, Robust, and Responsible AI*.

7.1 Introduction

While AI agents based on reinforcement learning (RL) (Sutton and A. G. Barto, 2018) have been remarkably successful in a variety of domains, ensuring the safety of agents developed using learning remains a major research challenge (Amodei et al., 2016; Hadfield-Menell et al., 2017; Orseau and Armstrong, 2016). Achieving safe and robust behaviour using state-of-the-art learning algorithms and a single

reward function is challenging (Leike et al., 2017). In simulated environments, agents are able to explore the whole action space, regardless of the consequences of their actions. However, in safety-critical applications where agents must learn in the real environment, such as autonomous cars (Mirchevska et al., 2018; Kendall et al., 2019), or chemical processes (Zhou et al., 2017; Savage et al., 2021), unconstrained exploration can lead to catastrophic failure, both during training and following deployment. In such cases, it is necessary to ensure that only safe actions are explored during learning. Such approaches are termed “*provably safe*” in (Krasowski, Thumm et al., 2023).

A number of approaches to provably safe RL have been proposed in the literature, including: *action replacement*, e.g., (Hunt et al., 2021), where unsafe actions are replaced by (default or sampled) safe ones; *action projection*, e.g., (Cheng et al., 2019), where unsafe actions are replaced by projecting them to the safe action space; and *action masking*, e.g., (Krasowski, Wang et al., 2020), where the agent is able to choose actions only from the safe action space. However, most provably safe approaches to RL are Markovian, that is, the set of actions which are considered “safe” is based only on the current state. There has been some work in which safety constraints are expressed in temporal logic. For example, *shields* (Alshiekh et al., 2018; ElSayed-Aly et al., 2021) is an action masking approach in which safety constraints are expressed in Safety LTL. The use of temporal logic means safety constraints can be specified over execution traces rather than states, allowing a much richer set of *non-Markovian safety constraints* to be used, e.g., “do not close the valve if it was opened within the last three seconds” (Alshiekh et al., 2018). However, generating a shield from a Safety LTL specification requires solving a safety game over an abstraction of the underlying MDP and is double exponential in the size of the Safety LTL specification. Moreover, the agent must learn in the resulting doubly exponential product MDP.

In this chapter, we present *Pure-Past Action Masking* (PPAM), a novel, lightweight approach to safe reinforcement learning. In PPAM, safety specifications are expressed in Pure Past Linear-time Temporal Logic (PPLTL) (De Giacomo, Di Stasio et al., 2020). PPLTL is a variant of Linear-time Temporal Logic in which formulas are evaluated over finite traces and only past modalities are allowed. PPLTL formulas have a very natural interpretation: the formula must be true at the end of the trace and is evaluated “backwards”. Many specifications about actions are easier and more natural to express when referring to the past (Lichtenstein et al., 1985), and past temporal logics have been used in closely related problems with non-Markovian aspects, e.g., non-Markovian models in reasoning about actions (Gabaldon, 2011), non-Markovian rewards in MDPs (Bacchus et al., 1996), and expressing norms in multi-agent systems (Alechina, Bulling et al., 2015). PPAM has the same expressive power as shields but has significantly lower computational complexity (linear overhead in the state size). Moreover, the features used in the safety constraint need not be the same as those used by the learning agent, allowing a clearer separation of concerns between safety constraints and reward specifications than with shields. Critically, we show that PPAM does not lead to any loss of generality, i.e., an agent trained with a mask is able to learn any optimal and safe (with respect to the safety constraints) policy in the MDP, if there is one. We illustrate our approach with a number of examples from the literature, and present experimental results which show that, in addition to ensuring

safety, PPAM can improve sample efficiency.

7.2 Preliminaries

In this section, we first present a slightly modified version of MDPs which is used in the rest of the chapter. This modified MDP assumes the existence of a set of *features* which are then used to define the set of states and of an “*action availability function*”, detailing which set of actions are available in any state. As we will see, the set of features used by our PPAMs need not be the same of the MDP, thus enabling a separation of concerns between the learning agent and the safety component in our problem. Following, we present an approach for safe reinforcement learning which is called “*shielding*” (Bloem, Könighofer et al., 2015). A shield can be used to enforce safety constraint on agents, both during training and execution of the agents. The application of shields to reinforcement learning was first proposed in (Alshiekh et al., 2018) and later extended to multi-agent domains in (ElSayed-Aly et al., 2021).

Definition 7.1 (Markov Decision Process). A (factorized) Markov Decision Process $M = \langle \mathcal{F}, S, \mu, A, \Lambda, T, R, \gamma \rangle$ is a tuple where:

- $\mathcal{F}$ is the set of (boolean) features of the MDP¹;
- $S = 2^{\mathcal{F}}$ is the set of states (corresponding to features evaluations);
- $\Lambda : S \rightarrow 2^A$ is the action availability function which maps each state of the MDP to the set of actions available in that state;
- $T : S \times A \rightarrow \Delta(S)$ is the transition function which, for a state s and an action a , returns a probability distribution for the next MDP state if $a \in \Lambda(s)$ and is otherwise undefined;
- and all the other elements are as in a vanilla MDP, (see Definition 2.1 in Section 2.1).

In this chapter, we consider also non-Markovian policies. A non-Markovian policy $\rho : S^+ \rightarrow Pr(Act)$, where S^+ is the set of all non-empty sequences made of states in S , is a function mapping sequences of states $\vec{s} = s_0 \dots s_n$ to probability distributions over actions, with the property that for any action $a \in A$ with $a \notin \Lambda(s_n)$ the probability of action a according to $\rho(\vec{s})$ is zero. The value of a policy is the expected sum of rewards obtained by an agent choosing actions according to the policy, discounted by the discount factor γ .

Shields are synthesised from Safety LTL specifications, allowing them to express non-Markovian constraints over traces. Safety LTL (Manna and Pnueli, 1995) is the fragment of LTL (Pnueli, 1977) in which only the X (“*next*”) and $\Box$ (“*always*”) modalities are allowed. Shields come in two variants: *post-posed* and *preemptive*. Post-posed shields act *after* the agent has chosen an action to perform. When a post-posed shield detects that the chosen action would or could lead to a constraint violation, it

¹For simplicity we consider only boolean features, but all our results extend to arbitrary features or even featureless MDPs.

modifies the action to a *safe* default one. Preemptive shields act *before* the agent has chosen an action to perform. A preemptive shield outputs, at each timestep, the set of actions it considers to be *safe*, from which the agent chooses one to perform. As PPAM is an action masking approach, we focus on preemptive shields, and we will refer to them as simply “shields” below.

Definition 7.2 (Shield (Alshiekh et al., 2018)). A (preemptive) shield is a tuple $\mathcal{S} = \langle Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda \rangle$ where:

- Q is the finite set of states;
- q_0 is the initial state;
- Σ_I is the input alphabet, where $\Sigma_I = \Sigma_I^1 \times \Sigma_I^2$;
- $\Sigma_O = 2^A$ is the output alphabet;
- $\delta : S \times \Sigma_I \rightarrow Q$ is the state transition function;
- $\lambda : Q \times \Sigma_I^1 \rightarrow \Sigma_O$ is the output function.

An MDP observer function $f : S \rightarrow L$ maps states of the underlying MDP to observations from a set L . The input alphabet Σ_I is given by $L \times A$, and $\Sigma_I^1 = L$.

A shield is built from a Safety LTL specification and an abstraction of the underlying MDP. The abstraction is a deterministic safety word automaton that rejects prefixes of sequences that do not correspond to possible runs of the underlying MDP (restricted to the relevant sets of features); essentially, it describes the physics of the system. Given an infinite run that is physically possible, it cycles through accepting states indefinitely. The Safety LTL specification is translated into a safety automaton (Kupferman and Vardi, 2001). A safety automaton is a tuple $\mathcal{A}_s = (Q_s, q_{0_s}, \Sigma_s, \delta_s, F_s)$, where Q_s is the set of states of the automaton, q_{0_s} the initial state, Σ_s the input alphabet, $\delta_s : Q_s \times \Sigma_s \rightarrow Q_s$ the transition function, and $F_s \subseteq Q_s$ the set of “safe” states. An infinite word over Σ_s is accepted by a safety automaton if and only if the run induced by it visits only safe states, i.e., states in F_s .

The product of the safety automaton and the abstract MDP can be seen as a infinite 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 restricts the action space available to the agent (through its output function λ) to that specified by the winning strategy.

Synthesising a shield requires time double exponential in the Safety LTL specification and linear in the size of the abstracted MDP. Given a Safety LTL formula φ , it is possible to build a safety automaton $\mathcal{A}_\varphi$ of size double exponential in the size of φ , such that $\tau, i \models \varphi$ iff $\tau_i \in L(\mathcal{A}_\varphi)$. Generating and solving the product game between the automaton and the abstracted MDP can be done in linear time through standard safety games-solving techniques (Mazala, 2002).

As an illustration, we briefly recall the “hot water tank” scenario from (Alshiekh et al., 2018).

Example 7.1. In the “hot water tank” scenario, the aim is to learn an energy-efficient controller for a hot water storage tank with a maximum capacity of 100 litres. A reward is associated with each water tank level, depending on how much energy is needed to keep the water in the tank hot. The outflow from the tank is always between 0 and 1 litres per second. The agent can open or close the tank intake valve. The inflow is between 1 and 2 litres per second when the valve is open. Whenever the valve is opened or closed, the setting must 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*”. The safety constraints can be expressed using the following Safety LTL formula:

$$\begin{aligned} & G (\text{level} > 0) \wedge G (\text{level} < 100) \wedge \\ & G ((\text{open} \wedge X \text{close}) \rightarrow XX \text{close} \wedge XXX \text{close}) \wedge \\ & G ((\text{close} \wedge X \text{open}) \rightarrow XX \text{open} \wedge XXX \text{open}) \end{aligned}$$

The first two conjuncts specify that the storage level should always be between 1 and 99 liters. The last two establish that whenever the inflow is switched off (respectively, on), then it remains so for the next three seconds.

7.3 Pure-Past Linear-time Temporal Logic

PPAM safety properties are expressed in Pure-Past Linear-time Temporal Logic (PPLTL) (De Giacomo, Di Stasio et al., 2020). As the name suggests, PPLTL is the past-time version of LTL on finite traces, in which formulas are evaluated on the last state in the history looking back towards the beginning of the trace. As a result, the paths on which formulas are evaluated are naturally finite.

Given a set of propositional symbols (or “fluents”) $Prop$, the formulas of PPLTL are defined by:

$$\varphi ::= p \in Prop \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid Y \varphi \mid \varphi S \varphi$$

where $p \in Prop$. $Y \varphi$ means “in the previous timestep, φ ” (‘Yesterday’), and $\varphi_1 S \varphi_2$ means “ φ_2 happened in the past, and since then φ_1 is true” (‘Since’). We also define the abbreviations $P \varphi$ as $\top S \varphi$, meaning “in some state in the past φ ” (‘Past’), and $H \varphi$ as $\neg P \neg\varphi$, meaning “in all states in the past φ ” (‘Historically’).

Formulas of PPLTL are interpreted over finite traces $\tau = \tau[0]\tau[1] \dots \tau[n-1]$ where $\tau[i]$ at instant/timestep i is a subset of $Prop$, intuitively representing the propositional symbols that are true at timestep i . With $length(\tau)$ we denote the length of τ . Given τ , a PPLTL formula φ , and instant i , the relation $\tau, i \models \varphi$ (φ holds at i in τ) is defined

inductively as follows:

$\tau, i \models p$	iff $p \in \tau[i]$
$\tau, i \models \tau$	always
$\tau, i \models \neg\varphi$	iff $\tau, i \not\models \varphi$
$\tau, i \models \varphi_1 \wedge \varphi_2$	iff $\tau, i \models \varphi_1$ and $\tau, i \models \varphi_2$
$\tau, i \models \varphi_1 \vee \varphi_2$	iff $\tau, i \models \varphi_1$ or $\tau, i \models \varphi_2$
$\tau, i \models \mathbf{Y} \varphi$	iff $\tau, i-1 \models \varphi$, with $i > 0$, or $i = 0$ and $\varphi \equiv \perp$, i.e., φ is a contradiction
$\tau, i \models \varphi_1 \mathbf{S} \varphi_2$	iff there exists k with $0 \leq k \leq i$ such that $\tau, k \models \varphi_2$ and for all j with $k < j \leq i$, $\tau, j \models \varphi_1$.

Given a PPLTL formula φ , we will write $\tau \models \varphi$ as an abbreviation of $\tau, \text{length}(\tau) \models \varphi$.

In order to evaluate a PPLTL formula φ on a trace τ , it is sufficient to know the truth values of the subformulas of φ in the current and previous timestep; formally:

Proposition 7.1. *Let φ be a PPLTL formula and $\text{Subf}(\varphi)$ 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 $\tau, n-1 \models \psi$ if and only if $\tau', n'-1 \models \psi$ for each $\psi \in \text{Subf}(\varphi)$. Then, it holds that $\tau, n \models \varphi$ if and only if $\tau', n' \models \varphi$.*

This claim follows from results in (Bonassi et al., 2023) which are stated in a somewhat different form.

Due to the fact that $\text{Subf}(\varphi)$ is linear in φ itself, the proposition above implies the following.

Observation 7.1. Evaluating a PPLTL formula φ on a trace τ can be done in time linear in $|\varphi|$ and constant in $\text{length}(\tau)$, given that the truth values of subformulas of φ in the preceding state are known.

Crucially, this allows us to evaluate the truth value of a PPLTL specification using only a set of truth values that is linear in size of the formula, regardless of how many timesteps have elapsed.

7.4 Pure-Past Action Masking

In this section, we introduce Pure-Past Action Masking (PPAM). In PPAM, a safety specification comprises a set of PPLTL formulas evaluated on the history which specify the set of actions allowed given the history. For each action $a \in A$, a PPAM mask contains a PPLTL formula φ_a over the mask's features (fluents) Prop , and the action a is available if and only if φ_a is true. Notice how there is no coupling between the set of features available to the agent and those available to the mask.

Definition 7.3 (PPAM mask). Given a (learning) agent acting in an MDP $M_{ag} = \langle \mathcal{F}, S, \mu, S^F, A, \Lambda, T, R, \gamma \rangle$, a PPAM mask $\langle \mathcal{L}, \{\varphi_a : a \in A\} \rangle$ is a pair where:

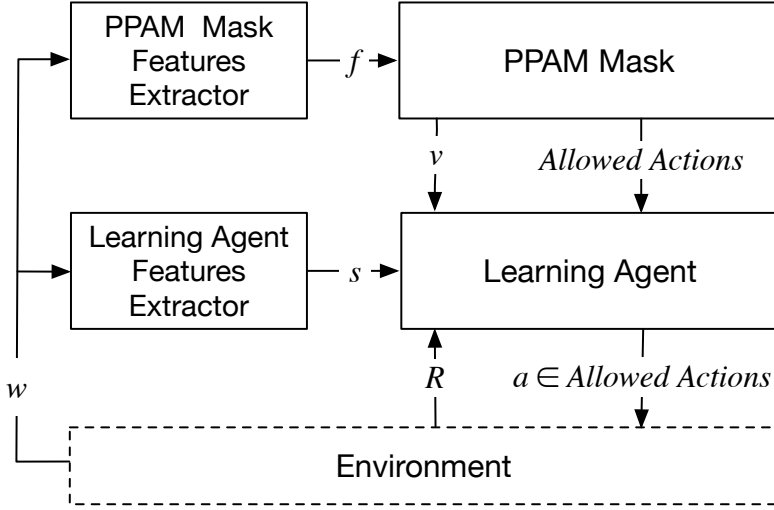


Figure 7.1: Interaction between the environment, agent and the PPAM mask. $Allowed\ Actions \subseteq \Lambda(s)$ is the set of permitted actions output by the PPAM mask, $v \in \mathcal{L}$ is the PPAM mask’s state, $a \in Allowed\ Actions$ is the action chosen by the agent, r is the reward given by the environment after a is performed, and w is the resulting state of the environment. From w , we obtain the MDP state s through the learning agent’s feature extractor, and f through the feature extractor of the PPAM mask (notice these need not be aligned, as $\mathcal{F}$ and $Prop$ are not necessarily equal).

- $\mathcal{L} = 2^{Prop}$ is the set of states of the mask;
- $\{\varphi_a : a \in A\}$ is a set containing a PPLTL formula φ_a over $Prop$ for each action $a \in A$, specifying when a is permitted.

As an example, we show how the safety specification for the “hot water tank” scenario can be expressed using PPAM action specifications.

Example 7.2. The “hot water tank” safety specification (see Example 7.1) can be expressed using the following PPLTL action specifications:

$$\varphi_{open} = level \leq 93 \wedge (close \rightarrow \mathbf{Y} \, close \wedge \mathbf{Y} \, \mathbf{Y} \, close)$$

$$\varphi_{close} = level \geq 4 \wedge (open \rightarrow \mathbf{Y} \, open \wedge \mathbf{Y} \, \mathbf{Y} \, open)$$

We now state the learning problem for PPAM.

Definition 7.4 (PPAM Learning Problem). An instance of an RL problem with a PPAM mask is a pair $M_{ag}^{ppam} = \langle M_{ag}, PPAM \rangle$ where $M_{ag} = \langle \mathcal{F}, S, \mu, S^F, A, \Lambda, T, R, \gamma \rangle$ is a factorized MDP with T and R hidden, and $PPAM = \langle \mathcal{L}, \{\varphi_a : a \in A\} \rangle$ is a PPAM mask with a set of PPLTL formulas φ_a over $Prop$, where $\mathcal{L} = 2^{Prop}$.

A solution to the problem is a non-Markovian policy $\rho : (S \times \mathcal{L})^+ \rightarrow \Delta(A)$ that maximises the expected discounted cumulative reward and conforms to the

PPAM mask's specification, that is, for every $\tau \in (S \times \mathcal{L})^+$, $\rho(\tau)(a) > 0$ only if $\tau' \models \varphi_a$ where τ' is τ projected on the second component (the sequence of sets of mask fluents).

Note that the solution is a policy in which all actions are safe (an action only has non-zero probability if the corresponding 'precondition' formula holds). In the following, we will assume that the PPAM learning problem is such that, for any possible trace that can be generated, there is always at least one action that can be taken, i.e., that the problem has at least one solution.

We now show how to solve the PPAM learning problem. We begin by lifting the evaluation of formulas on sequences of mask states from atomic fluents to all subformulas of mask formulas. By $\text{Subf}(\varphi)$ we denote the set of subformulas of φ (nodes in its parse tree). Let $V \subseteq 2^{\{\text{Subf}(\varphi_a) : a \in A\}}$ be the set of all possible truth assignments to $\{\text{Subf}(\varphi_a) : a \in A\}$. Given a sequence $\tau \in \mathcal{L}^+$, and the valuation of fluents in each state, it is possible to evaluate formulas from $v \in V$ on τ according to the truth definition for PPLTL formulas. In fact, as we have seen earlier, it is sufficient to know the valuation of $\{\text{Subf}(\varphi_a) : a \in A\}$ in the preceding state and the valuation of fluents in the current state in order to evaluate $\{\text{Subf}(\varphi_a) : a \in A\}$.

Note that each $v \in V$ determines the set of actions that are permitted given the mask's specification (those $a \in A$ where $\varphi_a \in v$, that is, φ_a is true). We are going to use this to reduce the non-Markovian PPAM learning problem to a Markovian learning problem.

Below we prove (Theorem 7.2) that the PPAM learning problem always has a Markovian solution $\pi : S \times V \rightarrow \Delta(A)$ that maximises the expected discounted cumulative reward and is safe (only actions permitted by the mask are used). The transition to a Markovian policy is possible because each $v \in V$ encodes sufficient information about the history of the system.

First, we define an important construction used in the proof of Theorem 7.2. It is an MDP that is essentially a product of the original MDP and the mask's states and transitions.

Definition 7.5 (Product MDP). Given an instance of a PPAM learning problem $M_{ag}^{ppam} = \langle M_{ag}, \text{PPAM} \rangle$ with $M_{ag} = \langle \mathcal{F}, S, \mu, S^F, A, \Lambda, T, R, \gamma \rangle$ and $\text{PPAM} = \langle \mathcal{L}, \{\varphi_a : a \in A\} \rangle$, we define a new product MDP $M_{ag \times ppam} = \langle \mathcal{F}', S', \mu', S^{F'}, A', \Lambda', T', R', \gamma' \rangle$, where

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

- $T' : (S \times v) \times A \rightarrow \Delta(S \times V)$ is defined as follows. As in (De Giacomo, Iocchi et al., 2019), we assume that there is a probability distribution $T^{ag \times ppam} : S \times \mathcal{L} \times A \rightarrow \Delta(S \times \mathcal{L})$ induced by the world (since the mask's fluents are caused by the agent acting in the world). This probability distribution is unknown to the agent, just as T is unknown. We lift the probability distribution $T^{ag \times ppam}$ to $T' : S \times V \times A \rightarrow \Delta(S \times V)$ as follows. The probability of transitioning from $\langle s, v \rangle$ to $\langle s', v' \rangle$ on executing action a is the same as the probability given by $T^{ag \times ppam}$ of transitioning from $\langle s, \ell \rangle$ to $\langle s', \ell' \rangle$ by a , where ℓ is the set of fluents in v , and v' is the maximal subset of $\{Subf(\varphi_a) : a \in A\}$ 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 7.1²;
- $R'(\langle s, v \rangle, a, \langle s', v' \rangle) := R(s, a, s')$;
- $\gamma' := \gamma$.

Note that the size of each state (s, v) is linear in the size of S and the size of the mask. The key feature of this product MDP is that only safe (permitted by the mask) actions are available to the agent. We prove in Theorem 7.2 below that learning on the product MDP with access to only the safe actions is sufficient for solving the PPAM learning problem. Critically, the optimality of learning with only safe actions is *entailed by the proof*: it is not part of the definition of the PPAM learning problem.

Theorem 7.2. *An RL problem with a PPAM mask $M_{ag}^{ppam} = \langle M_{ag}, PPAM \rangle$, with $M_{ag} = \langle \mathcal{F}, S, \mu, S^F, A, \Lambda, T, R, \gamma \rangle$ and $PPAM = \langle \mathcal{L}, \{\varphi_a : a \in A\} \rangle$, can be reduced to RL over the product MDP $M_{ag \times ppam}$ such that optimal non-Markovian policies for M_{ag}^{ppam} can be learned by learning corresponding optimal Markovian policies for $M_{ag \times ppam}$.*

Proof. First, observe that the actions available to the agent in M_{ag}^{ppam} after each $\langle s_0, \ell_0 \rangle \dots \langle s_n, \ell_n \rangle$ (where ℓ_i are sets of fluents at timestep i) are the same as the actions available to the agent in the MDP $M_{ag \times ppam}$ in $\langle s_n, v_n \rangle$ where v_n is the set of $\psi \in \{Subf(\varphi_a) : a \in A\}$ such that $\ell_0 \dots \ell_n \models \psi$. This is because the evaluation of formulas from $\{Subf(\varphi_a) : a \in A\}$ including φ_a for each $a \in A$ is the same in $\langle s_0, \ell_0 \rangle \dots \langle s_n, \ell_n \rangle$ and in $\langle s_n, v_n \rangle$. Hence, instead of masking actions based on $\langle s_0, \ell_0 \rangle \dots \langle s_n, \ell_n \rangle$, we can mask them depending on $\langle s_n, v_n \rangle$ without loss of information. Note that the rewards are the same in both MDPs. This means that the reward for performing a after $\langle s_0, \ell_0 \rangle \dots \langle s_n, \ell_n \rangle$ is the same as the reward for performing a in $\langle s, v_n \rangle$. Since $T' : S \times V \times A \rightarrow \Delta(S \times V)$ corresponds to $T^{ag \times ppam} : S \times \mathcal{L} \times A \rightarrow \Delta(S \times \mathcal{L})$, the optimal non-Markovian policy $\rho^* : (S \times \mathcal{L})^+ \rightarrow \Delta(A)$ produces exactly the same reward as a Markovian policy $\pi : S \times v \rightarrow \Delta(A)$ obtained by replacing $\ell_0, \dots, \ell_n$ with the corresponding v_n . $\square$

Note that, by construction, the learned policy is guaranteed to conform to the PPAM mask specification (since it is learned in the product MDP $M_{ag \times ppam}$ where only safe actions are available) and to be optimal (among policies that conform to the PPAM mask specifications). In other words, this result tells us that PPAM masks

²Observe 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 7.1.

do not restrict agents from finding optimal (safe) policies for the original learning problem, while at the same time automatically guaranteeing safety.

Moreover, the input to the learning algorithm is the product MDP $M_{ag \times ppam}$. In the product MDP, the number of new features added to the states is linear with respect to the size of the mask's specification, as the new feature space is exactly the union of $\mathcal{F}$ (the feature space of M_{ag}) and $\{Subf(\varphi_a) : a \in A\}$. This leads to the following corollary regarding the complexity of using PPAM.

Corollary 7.3. *RL with a PPAM mask incurs a single exponential, in the size of the mask specification, overhead in the size of the state space. Moreover, if the MDP is factorized, a PPAM mask leads to a linear, again in the size of the mask specification, overhead in the size of the feature space.*

In practice, the increase in the size of the state space may be lower than the single exponential blowup in Corollary 7.3, as the truth value of some subformulas may imply that of other subformulas.

7.5 Comparison with Shields

In this section, we briefly compare the expressive power of PPAM and shields. Such a comparison is not entirely straightforward. Shields are built using an abstraction of the underlying MDP that describes the dynamics of the learning environment. In contrast, the set of fluents used to specify PPAM safety constraints are independent of the features of the MDP seen by the learning agent. For the comparison to be meaningful, we must assume that the states in the abstract MDP used to build the shield are a subset of the fluents available to the PPAM.

Let $\mathcal{S} = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$ be a shield synthesised from some safety automaton (Safety LTL specification) $\mathcal{A}_s$, where $Q = Q_M \times Q_s$ is the set of states, Q_M is the set of states of the MDP abstraction, Q_s is the sets of states of the safety automaton used to synthesise $\mathcal{S}$, and $\Sigma_I = L \times A$.

First, we prove a lemma connecting DFAs, built from (Safety) LTL formulas, and PPLTL.

Lemma 7.4. *For any state q_s of the safety automaton $\mathcal{A}_s$ used to synthesise the shield $\mathcal{S}$, it is possible to provide a PPLTL formula ϕ_{q_s} describing exactly the paths leading to q_s from the initial state of $\mathcal{A}_s$.*

Proof. As the safety automaton $\mathcal{A}_s$ is synthesised from a (Safety) LTL formula, it must be counter-free (Manna and Pnueli, 1990). The claim then follows via a result from (Zuck, 1986), showing that for any given counter-free automaton it is possible to build a PPLTL formula ϕ_q recognizing all paths leading to any of its states q . $\square$

Notice that, if the MDP abstraction is a counter-free automaton, then we could apply Lemma 7.4 to obtain a PPLTL formula for each of its states q_M . For generality, we assume that it is an arbitrary DFA and include the set of states Q_M of the MDP abstraction amongst the fluents that we can use to define our PPLTL formulas.

The next lemma is crucial for stating our expressiveness result. We assume that we have access to the state of the MDP abstraction, the observation from f of L and

the action executed by the agent, i.e., that the set of fluents is $Prop = Q_M \cup L \cup A$.³ We show how, for each action $a \in A$, we can build a PPLTL formula φ_a that will constrain a in exactly the same way as S .

Lemma 7.5. *Given a shield S , for any action $a \in A$ we can build a PPLTL formula φ_a over fluents $Prop = Q_M \cup L \cup A$ such that, for any shield state (q_M, q_s) and observation ι reached by S after some trace τ , $a \in \lambda(\langle q_M, q_s \rangle, \iota)$ if and only if $\tau \models \varphi_a$.*

Proof. First, let $a \in A$ be an arbitrary action. If an MDP abstract state-observation pair (q_M, Σ_I) is such that $a \in \lambda(\langle q_M, q_s \rangle, \iota)$ for any safety automaton state q_s (reachable from the initial state q_0) and for which ι is observable from (q_M, q_s) , then q_M and ι being the current MDP abstraction state and observation is a sufficient precondition to allow the agent to take action a . Thus, we define Θ^a as the set of all such MDP abstraction state-observation pairs.

Now, let q_M be an MDP abstraction state and ι an observation such that there are two distinct safety automaton states q_s, q'_s for which: (i) both q_s and q'_s are reachable from q_0 , (ii), ι can be observed in both (q_M, q_s) and (q_M, q'_s) , and (iii), $a \in \lambda(\langle q_M, q_s \rangle, \iota)$ and $a \notin \lambda(\langle q_M, q'_s \rangle, \Sigma_I)$. For q_M and ι , we need to keep track of the current state q_s in the safety automaton, so that we allow a if and only if the current state q_s is such that $a \in \lambda(\langle q_M, q_s \rangle, \iota)$. Let $Q_s^{a, q_M, \iota} = \{q_s \in Q_s : a \in \lambda(\langle q_M, q_s \rangle, \iota)\}$. Thanks to Lemma 7.4, we can build for each of these $q_s \in Q_s^{a, q_M, \iota}$ a PPLTL formula ϕ_{q_s} that is satisfied if and only if, given the current history, the safety automaton $\mathcal{A}_s$ is in state q_s .⁴

We are now ready to define the PPLTL formula φ_a that will constrain action a . In the formula we implicitly omit all the $\langle q_M, \iota \rangle \in (Q_M \times L) \setminus \Theta^a$ such that $Q_s^{a, q_M, L} = \emptyset$.

$$\varphi_a = \bigvee_{\langle q_M, \iota \rangle \in \Theta^a} (q_M \wedge \iota) \vee \bigvee_{\langle q_M, \iota \rangle \in (Q_M \times L) \setminus \Theta^a} \left(q_M \wedge \iota \wedge \left(\bigvee_{q_s \in Q_s^{a, q_M, \iota}} \phi_{q_s} \right) \right)$$

We now show that action a is allowed by the shield S if and only if the current trace τ satisfies φ_a . Let $\langle q_M, q_s \rangle$ be the current state of the shield and ι the current observation, reached after following the trace τ . Observe that for $a \in \lambda(\langle q_M, q_s \rangle, \iota)$, we either have that $a \in \lambda(\langle q_M, q'_s \rangle, \iota)$ for any state of the safety automaton q'_s such that $\langle q_M, q'_s \rangle$ is reachable from $q_{0,S}$ and ι is observable at $\langle q_M, q'_s \rangle$, or not.

In the first case, it means that whenever the state of the MDP abstraction is q_M and the observation is ι , the shield will always allow action a . Notice that the fluent for state q_M and for observation ι are true in the last timestep of trace τ if and only if q_M is the current state of the MDP abstraction and ι the current observation. Hence, S allows action a if and only if $\tau \models \varphi_a$, since $\tau \models q_M \wedge \iota$ for $\langle q_M, \iota \rangle \in \Theta^a$.

In the second case, it means that the current safety automaton state q_s is such that, when coupled with the current MDP abstraction state q_M and observation ι ,

³We implicitly assume that finite traces we consider when evaluating the formula φ_a will be of the form $\langle q_M^0, \iota^0 \rangle a^0 \langle q_M^1, \iota^1 \rangle a^1 \dots \langle q_M^n, \iota^n \rangle$, where a^i is the action taken after $\langle q_M^i, \iota^i \rangle$.

⁴Notice that the safety automaton transitions on inputs from $L \times A$, hence we also need A to be included in the set of fluents available to define the PPLTL formula φ_a .

$a \in \lambda(\langle q_M, q_s \rangle, \iota)$. This is true if and only if the trace τ is such that: (i) q_M and ι are true at its final timestep, and (ii), $\tau \models \phi_{q_s}$, by Lemma 7.4. Hence, we have again that $\mathcal{S}$ allows action a if and only if $\tau \models \varphi_a$, since $\tau \models q_M \wedge \iota \wedge \left(\bigvee_{q_s \in Q_s^{a, q_M}} \phi_{q_s} \right)$ for $\langle q_M, \iota \rangle \in (Q_M \times L) \setminus \Theta^a$ and $q_s \in Q_s^{a, q_M, \ell}$. $\square$

Theorem 7.6. *For every shield $\mathcal{S}$ for a safety property φ , there exists a PPAM mask over fluents $Prop = Q_M \cup L \cup A$ such that the policy learned with it will satisfy φ .*

Proof. For each action $a \in A$, we build the formula φ_a from Lemma 7.5. Then, it is easy to see that the resulting PPAM mask constrains actions just like the input shield $\mathcal{S}$ does. Therefore, the agent learns exactly the same policy. $\square$

Theorem 7.6 shows that, when provided with the same information, PPAM masks have the same expressive power as shields in constraining actions.

We conclude this section by briefly discussing some practical considerations relating to the use of PPAMs and shields. PPAMs require that the conditions for the safe execution of each action are specified directly by the designer of the mask. In contrast, with shields, the safety property is specified at system behavior level, and the constraints on actions are synthesised during the construction of the shield. While this avoids the designer of the shield having to specify action conditions, construction of the shield requires that an abstraction of the underlying MDP is available (which may be challenging for cyber-physical systems that learn in the real environment), and entails significant computational overhead compared to PPAM masks. PPAMs are therefore more appropriate in situations where there is a straightforward correspondence between safety specifications and actions, and/or an abstraction of the underlying MDP is unavailable. Another clear difference between the two approaches follows from the logics that they employ, i.e., Safety LTL and PPLTL. Safety LTL has only *future* modalities, and as result, shields are more appropriate when the specification constrains how the system can evolve in the future. On the other hand, PPLTL (as the name suggests) has only *past* modalities, and so PPAMs are more appropriate when the action constraints depend on what has happened in the past.

7.6 Experimental Evaluation

In this section, we illustrate our approach with two examples from the literature, COCKTAILPARTY (De Giacomo, Iocchi et al., 2019) and BOATRACE (Leike et al., 2017), and present experimental results which show that, in addition to ensuring safety, PPAMs can also improve sample efficiency. Code and more details about the environments is available in a GitHub repository at: github.com/giovannivarr/PPAM-AAAI24/. The repository also includes a PPAM implementation of the WATERTANK environment from (Alshiekh et al., 2018). We have not included the WATERTANK results here, as they follow directly from Theorem 7.6 in the previous section: both the PPAM- and shield-based agents are constrained in the same way, and learn exactly the same policies. All experiments were performed on an Apple M1 chip with a 16 GB RAM, macOS Ventura 13.5 laptop.

7.6.1 COCKTAILPARTY

In the COCKTAILPARTY scenario the agent learns with a *restraining bolt* (RB), which provides additional rewards to the agent depending on whether the safety specification is satisfied. However, restraining bolts do not provide any safety guarantees, either during training or deployment. In the scenario, the agent is a robot which serves drinks (coke and beer) and snacks (biscuits and chips) to four guests at a party. The environment is a 5×5 grid. The robot knows the locations of drinks and snacks and the locations of people, and can execute actions to move in the environment, grasp items and serve them to people. The robot gets a reward when a delivery task is completed. Some of the guests are children who should not be served alcohol, and the robot should not serve more than one drink or snack to each guest. However, 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).

To enforce the constraint that children should not be served alcohol and each guest should be served one drink or snack, we use a PPAM mask to constrain the “serve” action, using the following PPLTL formula:

$$\begin{aligned} \varphi_{\text{serve}} = & \bigvee_{g \in \text{Guests}} [at_g \wedge \\ & ((\neg P \text{ served_food_}g \wedge \text{holding_food}) \vee \\ & (\neg P \text{ served_drink_}g \wedge \text{holding_drink} \wedge \\ & (\neg \text{minor_}g \vee \neg \text{holding_alcohol})))] \end{aligned}$$

where at_g is true if the robot is in the cell occupied by guest g , $holding_food$ is true if the robot is holding a snack in the current timestep, $holding_drink$ is true if the robot is holding a drink, $holding_alcohol$ is true if the robot is holding an alcoholic drink, $minor_g$ is true if guest g is a minor, $served_food_g$ is true if guest g has been served food previously, and $served_drink_g$ is true if the guest has been served a drink previously.

We use the same experimental setup as in (De Giacomo, Iocchi et al., 2019): both agents are trained using n-step Sarsa, configured with learning rate $\alpha = 0.999$, exploration rate $\epsilon = 0.2$, and $n = 100$. We performed 10 runs, each consisting of 4000 iterations lasting at most 1000 steps. Each run is split into non-overlapping intervals containing 100 consecutive iterations. For each such interval, we evaluate the average reward obtained by the agent. Figure 7.2 shows the median average rewards per interval and the shaded areas enclose the 25th and 75th percentiles. As can be seen, the PRB agent achieves a higher median average reward and learns to complete all four serving tasks more quickly than the RB agent. This is consistent with the observation in (Shengyi Huang and Onta  n, 2022), that the improvement in sample efficiency is due to the fact that the PPAM agent has fewer actions to choose from. Moreover, the median number of violations of the safety specification by the RB agent was 164.5.

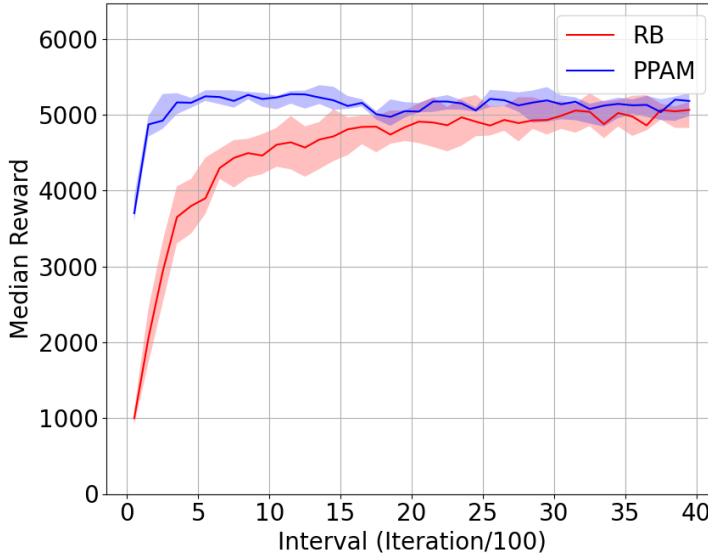


Figure 7.2: Median reward of PPAM and RB agents.

7.6.2 BOATRACE

BOATRACE is an “AI Safety Gridworld” (Leike et al., 2017) designed to illustrate ‘reward gaming’ (Skalse et al., 2022), a general phenomenon where an agent exploits an unintended loophole in the reward specification to obtain rewards in unintended (and possibly unsafe) ways. The environment is a 5×5 grid, and the agent sails around a track, going past 4 different checkpoints, one for each cardinal direction. The agent should learn to complete as many clockwise laps as possible (performance wrt the goal is measured as the number of clockwise movements minus the number of counter-clockwise movements the agent has taken across the entirety of the episode). The agent receives a reward of -1 for each step taken, and, if it arrives on a checkpoint from the “correct” clockwise direction, it also receives a reward of 3. As a result, the agent can maximize its reward by stepping back and forth on a checkpoint. This is not the desired behavior and may be unsafe, e.g., if there are other agents in the environment.

To ensure correct behavior, we use a PPAM that constrains the agent’s movement actions. The PPLTL formula constraining the “move_{west}” action is:

$$\varphi_{\text{move}_{\text{west}}} = (\neg(\text{north} \vee \text{west})) \text{ S } (\text{south} \vee \text{east})$$

where the fluent corresponding to each cardinal direction is true whenever the agent moves onto the corresponding checkpoint in the track. The constraint checks that the last checkpoint that was visited was either east or south, as after these the agent can move west. The PPLTL formulas for the other movement actions are analogous:

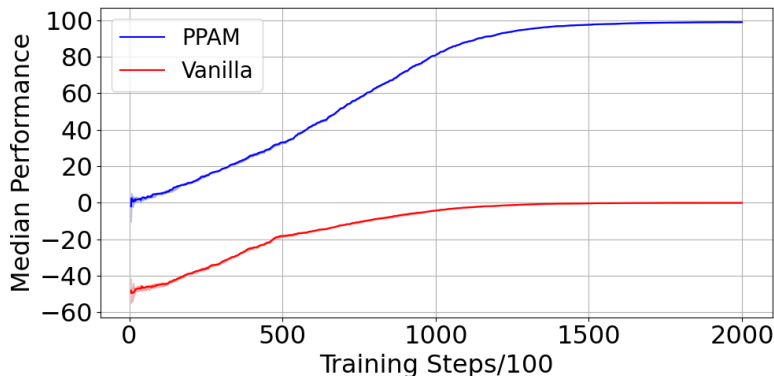


Figure 7.3: Median performance of PPAM and Q agents.

the agent is allowed to perform the `move_north` action only if the last checkpoint visited was either west or south; the agent is allowed to perform the `move_south` action only if the last checkpoint was either north or east, and the agent is allowed to perform the `move_east` action, only if the last checkpoint was either north or west.

We compare a Q-learning agent trained with a PPAM to a vanilla Q-learning agent. Both agents are configured with learning rate $\alpha = 0.5$, and initial exploration rate $\epsilon = 1$, linearly annealed to 0.01 over 100000 training steps. We performed 10 runs, each consisting of 200000 training steps. The maximum performance obtainable is 100. Lines in Figure 7.3 show the median performance during training.

As can be seen, the PPAM-based agent learns the optimal policy with respect to the performance function, i.e., it learns how to correctly complete laps. In contrast, the vanilla agent achieves a performance score of 0, meaning that it performs as many clockwise and counter-clockwise movements: in other words, it has learned to exploit the reward function and moves back and forth over a single checkpoint. Note that the poor performance of the non-PPAM agent is not due to the use of Q-learning. In (Leike et al., 2017) agents were trained using deep RL algorithms (Rainbow (Hessel et al., 2018) and A2C (Mnih et al., 2016)), but did not manage to learn a policy that optimizes performance in this environment.

7.7 Conclusions

We have presented *pure-past action masking*: a lightweight, modular formalism for specifying safe or permitted actions of reinforcement learning agents. The mask specification comprises a set of PPLTL formulas evaluated on the history, and specifies the set of actions permitted in the current state. As formulas are evaluated on the history rather than just the current state, PPAM enables non-Markovian action masking. The PPAM specification is enforced by construction: a PPAM agent will always learn an optimal policy which never violates its safety specification during either training or execution. PPAM can be applied in any setting where the fluents

relevant to the safety specification can be perceived by the mask, and where the conditions on action availability can be expressed in PPLTL, which is as expressive as Safety LTL. Moreover, using PPAM incurs only a single exponential blowup with respect to the size of the PPLTL formula.

In future work, we plan to extend the framework of PPAM to multi-agent reinforcement learning, as has been done with shields (ElSayed-Aly et al., 2021). We would also like to investigate possible ways of implementing PPAM in environments with continuous action spaces, as in, e.g., (Krasowski, Thumm et al., 2023). Finally, another interesting extension would be to employ PPAMs in partially observable MDPs (POMDPs). As PPAMs make no assumptions about the underlying structure of the (PO)MDP, if the PPAM's fluents are fully observable, then this trivially reduces to the approach presented in this chapter. If the PPAM's fluents are not full observable, we can build on work on evaluating temporal logic formulas on incomplete traces, e.g., (Joshi et al., 2017).

7.8 Supplementary Material

Appendix A: Proof of Proposition 1

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;
- $\varphi = Y \psi$. Then, $\tau, n \models Y \psi$ iff $\tau, n-1 \models \psi$ iff, since $\psi \in Subf(Y \psi)$, $\tau', n'-1 \models \psi$ iff $\tau', n' \models Y \psi$;
- $\varphi = \psi_1 S \psi_2$ and consider the one-step unfolding equivalence $\psi_1 S \psi_2 \equiv \psi_2 \vee (\psi_1 \wedge Y \psi_1 S \psi_2)$. Then, we have that $\tau, n \models \psi_1 S \psi_2$ iff $\tau, n \models \psi_2 \vee (\psi_1 \wedge Y \psi_1 S \psi_2)$ iff $\tau, n \models \psi_2$ or $\tau, n \models \psi_1 \wedge Y \psi_1 S \psi_2$ and we distinguish two cases:
 - $\tau, n \models \psi_2$, 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 $\tau', n' \models \psi_1 S \psi_2$;
 - $\tau, n \models \psi_1 \wedge Y \psi_1 S \psi_2$ iff $\tau, n \models \psi_1$ and $\tau, n \models Y \psi_1 S \psi_2$. On the one hand, from $\tau, n \models \psi_1$ we easily obtain that $\tau', n' \models \psi_1$. On the other hand, from $\tau, n \models Y \psi_1 S \psi_2$ it follows that $\tau, n-1 \models \psi_1 S \psi_2$. Now, since $\psi_1 S \psi_2 \in Subf(\varphi)$, it holds that $\tau', n'-1 \models \psi_1 S \psi_2$ and so that $\tau', n' \models Y \psi_1 S \psi_2$. We therefore obtain $\tau', n' \models \psi_1 \wedge Y \psi_1 S \psi_2$, which implies $\tau', n' \models \psi_1 S \psi_2$.

□

Appendix B: Proof of Theorem 1

Proof. First, observe that the actions available to the agent in M_{ag}^{ppam} after each $\langle s_0, \ell_0 \rangle \dots \langle s_n, \ell_n \rangle$ (where ℓ_i are sets of fluents at timestep i) are the same as the actions available to the agent in the MDP $M_{ag \times ppam}$ in $\langle s_n, v_n \rangle$ where v_n is the set of $\psi \in \{Subf(\varphi_a) : a \in A\}$ such that $\ell_0 \dots \ell_n \models \psi$. This is because the evaluation of formulas from $\{Subf(\varphi_a) : a \in A\}$ including φ_a for each $a \in A$ is the same in $\langle s_0, \ell_0 \rangle \dots \langle s_n, \ell_n \rangle$ and in $\langle s_n, v_n \rangle$. Hence, instead of masking actions based on $\langle s_0, \ell_0 \rangle \dots \langle s_n, \ell_n \rangle$, we can mask them depending on $\langle s_n, v_n \rangle$ without loss of information. Note that the rewards are the same in both MDPs. This means that the reward for performing a after $\langle s_0, \ell_0 \rangle \dots \langle s_n, \ell_n \rangle$ is the same as the reward for performing a in $\langle s_n, v_n \rangle$. Since $T' : S \times V \times A \rightarrow \Delta(S \times V)$ corresponds to $T^{ag \times ppam} : S \times \mathcal{L} \times A \rightarrow \Delta(S \times \mathcal{L})$, the optimal non-Markovian policy $\rho^* : (S \times \mathcal{L})^+ \rightarrow \Delta(A)$ produces exactly the same reward as a Markovian policy $\pi : S \times v \rightarrow \Delta(A)$ obtained by replacing $\ell_0, \dots, \ell_n$ with the corresponding v_n . $\square$

Appendix C: Proof of Lemma 2

In the following, we denote by Q_M and Q_s the set of states of, respectively, the MDP abstraction and safety automaton that were used to synthesise the input shield $\mathcal{S}$; L is the set of input observations for the shield, we denote with ι elements of L .

Proof. First, let $a \in A$ be an arbitrary action. If an MDP abstract state-observation pair $\langle q_M, \iota \rangle$ is such that $a \in \lambda(\langle q_M, q_s \rangle, \iota)$ for any safety automaton state q_s (reachable from the initial state q_0) and for which ι is observable from $\langle q_M, q_s \rangle$, then q_M and ι being the current MDP abstraction state and observation is a sufficient precondition to allow the agent to take action a . Thus, we define Λ^a as the set of all such MDP abstraction state-observation pairs.

Now, let q_M be an MDP abstraction state and ι an observation such that there are two distinct safety automaton states q_s, q'_s for which: (i) both q_s and q'_s are reachable from q_0 , (ii), ι can be observed in both $\langle q_M, q_s \rangle$ and $\langle q_M, q'_s \rangle$, and (iii), $a \in \lambda(\langle q_M, q_s \rangle, \iota)$ and $a \notin \lambda(\langle q_M, q'_s \rangle, \iota)$. For q_M and ι , we need to keep track of the current state q_s in the safety automaton, so that we allow a if and only if the current state q_s is such that $a \in \lambda(\langle q_M, q_s \rangle, \iota)$. Let $Q_s^{a, q_M, \iota} = \{q_s \in Q_s : a \in \lambda(\langle q_M, q_s \rangle, \iota)\}$. Thanks to Lemma 7.4, we can build for each of these $q_s \in Q_s^{a, q_M, \iota}$ a PPLTL formula ϕ_{q_s} that is satisfied if and only if, given the current history, the safety automaton $\mathcal{A}_s$ is in state q_s ⁵.

We are now ready to define the PPLTL formula φ_a that will constrain action a . In the formula we implicitly omit all the $\langle q_M, \iota \rangle \in (Q_M \times L) \setminus \Theta^a$ such that $Q_s^{a, q_M, \iota} = \emptyset$.

$$\varphi_a = \bigvee_{\langle q_M, \iota \rangle \in \Theta^a} (q_M \wedge \iota) \vee \bigvee_{\langle q_M, \iota \rangle \in (Q_M \times L) \setminus \Theta^a} \left(q_M \wedge \iota \wedge \left(\bigvee_{q_s \in Q_s^{a, q_M, \iota}} \phi_{q_s} \right) \right)$$

⁵Notice that the safety automaton transitions on inputs from $L \times A$, hence why we also need A to be included in the set of fluents available to define the PPLTL formula φ_a .

We now show that action a is allowed by the shield $\mathcal{S}$ if and only if the current trace τ satisfies φ_a . Let $\langle q_M, q_s \rangle$ be the current state of the shield and ι the current observation, reached after following the trace τ . Observe that $a \in \lambda(\langle q_M, q_s \rangle, \iota)$ if and only if, for any state of the safety automaton q'_s such that $\langle q_M, q'_s \rangle$ is reachable from the shield's initial state q_0 and ι is observable at $\langle q_M, q'_s \rangle$, either $a \in \lambda(\langle q_M, q'_s \rangle, \iota)$ or not.

In the first case, it means that whenever the state of the MDP abstraction is q_M and the observation is ι , the shield will always allow action a . Notice that the fluent for state q_M and for observation ι are true in the last timestep of trace τ if and only if q_M is the current state of the MDP abstraction and ι the current observation. Hence, $\mathcal{S}$ allows action a if and only if $\tau \models \varphi_a$, since $\tau \models q_M \wedge \iota$ for $\langle q_M, \iota \rangle \in \Theta^a$.

In the second case, it means that the current safety automaton state q_s is such that, when coupled with the current MDP abstraction state q_M and observation ι , $a \in \lambda(\langle q_M, q_s \rangle, \iota)$. This is true if and only if the trace τ is such that: (i) q_M and ι are true at its final timestep, and (ii), $\tau \models \phi_{q_s}$, by Lemma 7.4. Hence, we have again that $\mathcal{S}$ allows action a if and only if $\tau \models \varphi_a$, since $\tau \models q_M \wedge \iota \wedge \left(\bigvee_{q_s \in Q_s^{a, q_M}} \phi_{q_s} \right)$ for $(q_M, \iota) \in (Q_M \times L) \setminus \Theta^a$ and $q_s \in Q_s^{a, q_M, \iota}$. $\square$

Appendix D: Figures of the Environments

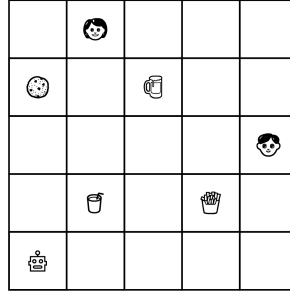


Figure 7.4: COCKTAILPARTY environment (De Giacomo, Iocchi et al., 2019).

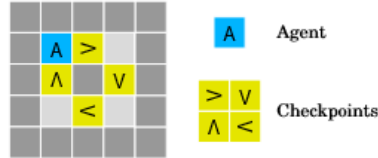


Figure 7.5: BOATRACE environments (Leike et al., 2017). Dark grey tiles represent walls.

Appendix E: Further Explanation of BOATRACE

The PPLTL formulas for the `move_west`, `move_north`, `move_south` and `move_east` actions are as follows:

$$\varphi_{\text{move_west}} = (\neg(\text{north} \vee \text{west})) \text{ S } (\text{south} \vee \text{east})$$

$$\varphi_{\text{move_north}} = (\neg(\text{north} \vee \text{east})) \text{ S } (\text{west} \vee \text{south})$$

$$\varphi_{\text{move_south}} = (\neg(\text{south} \vee \text{west})) \text{ S } (\text{north} \vee \text{east})$$

$$\varphi_{\text{move_east}} = (\neg(\text{south} \vee \text{east})) \text{ S } (\text{north} \vee \text{west})$$



8 Conclusions and Future Directions

In this dissertation, we have specifically focused on applying approaches and tools from declarative AI, and its related fields such as mathematics and formal methods, in reinforcement learning. We have seen how can these approaches be used to improve safety, training speed, and the quality of the resulting agents' policies. In this final chapter, we provide an overview of the contributions and outline possible directions that can be pursued in future research.

8.1 Contributions of the Dissertation

In this section we summarize the novel contributions presented in this work, which are included in Chapter 4, Chapter 5, Chapter 6, and Chapter 7, and relate them to other contributions in the field.

We first recall the thesis of this dissertation, as presented in the Introduction:

Thesis. *Declarative AI techniques can be leveraged in reinforcement learning to: (i) improve efficiency and quality of resulting policies for non-Markovian tasks, and (ii), guarantee the satisfaction of safety constraints both at training and execution time.*

In this dissertation, we have provided several approaches in support of this thesis. We have leveraged reward machines, combined with tools and techniques from declarative AI, to address point (i). We have presented two approaches to synthesise reward machines, one that uses planning, and the other that uses Alternating-time Temporal Logic. We have then defined pushdown reward machines by equipping reward machines with a pushdown stack. Pushdown reward machines are more expressive than reward machines, and can thus be used to train agents in achieving more complex tasks. Finally, we have addressed point (ii) by leveraging Pure-Past Linear-time Temporal Logic (PPLTL). We have constrained the agent to be able to perform an action if and only if a corresponding PPLTL formula is currently true. As the agent is not allowed to select actions that violate safety constraints, our approach guarantees that the agent will never violate any safety constraint, even during training.

This dissertation places itself in growing parts of the reinforcement learning literature. Several works have used reward machines to improve training efficiency of RL agents. We have first leveraged planning to obtain reward machines to train agents in achieving tasks in versatile ways. Our approach contributed to other works, e.g., (Illanes et al., 2019; Trapasso and A. Jonsson, 2025), which showed how reward machines can be synthesised using planning. Combined with other works that synthesise reward machines from a variety of specifications, e.g., natural-language prompts for large language models (Alsadat, Gaglione et al., 2025; Castanyer et al., 2025), our contribution further establishes reward machines as a tool that can be easily used to train agents in achieving non-Markovian tasks. We also contributed to research in multi-agent RL by devising a new approach that uses reward machines in cooperative MARL. Our contribution is, in this case, twofold. First, we provided a new approach to train agents in achieving non-Markovian tasks in a cooperative multi-agent setting. This is still a relatively new area, as only works that use reward machines in MARL (see Section 3.1), and (Eappen and Jagannathan, 2023; Alsadat and Xu, 2024) seem to provide an approach to do so. Secondly, we provided one of the first methods to synthesise reward machines for MARL from formal specifications. Up to our knowledge, this is done only in our work and in (C. Zhu et al., 2024; Trapasso and A. Jonsson, 2025). There have also been several works which try to augment the expressivity or modify the definition of reward machines, e.g., (Bester et al., 2024; Furelos-Blanco et al., 2023; Zheng and Yu, 2024; Ardon, Furelos-Blanco, Parac et al., 2025). From this point of view, the pushdown reward machines presented in this dissertation closed a research direction that was explicitly mentioned in the expanded work that introduced reward machines (Toro Icarte, Klassen, Valenzano and McIlraith, 2022). Finally, this dissertation contributed also to *provably safe* RL, another growing subfield of RL. After the publication of (Krasowski, Thumm et al., 2023), several other works in provably safe RL have followed, e.g., (Banerjee et al., 2024; Stolz et al., 2024; Brorholt et al., 2025). The approach we proposed in this dissertation currently stands out as the only one employing PPLTL. It would thus be interesting to explore possible synergies of our past-based approach with the newer works in provably safe RL.

We will now give, for each chapter, a short summary including the research question it aimed to address and possible future research directions.

8.1.1 Chapter 4: Synthesis of Maximally Permissive Reward Machines

We have first explored in Chapter 4 the synthesis of what we called *maximally permissive* reward machines. In this work, the main objective was to train agents in achieving tasks in all the possible ways that they could be achieved, hence the “maximally permissive”. We did this by employing planning, inspired by the previous work of Illanes et al. (2019), where reward machines were obtained from an individual sequential or partial-order plan.

In our approach we assume that the MDP (in which the agent acts and learn) and the task the agent is learning are abstracted into, respectively, a planning domain and a planning task. From the given planning domain and the planning task we obtain all partially-ordered plans that achieve the task: using these, we can then

synthesise the maximally permissive reward machine to train the agent with. We have established theoretically that agents trained with maximally permissive reward machines will always converge to policies which accrue an expected discounted reward that is greater than or equal to that of policies learned by agents that are trained with reward machines obtained from single sequential plans or single partial-order plans. In the experimental evaluation we have seen in practice how different levels of *permissiveness* can affect training and the resulting policy: while the agents trained with the maximally permissive reward machines always managed to achieve optimal behaviours, this came to the cost of a slower convergence speed in training. We hypothesised that the slower convergence was due to the larger state spaces of the maximally permissive reward machines, as the agents needed more time to explore all the possible ways of achieving the tasks and to find the optimal one out of all of them.

There are two possible ways to try to address the slower convergence speed caused by maximally permissive reward machines. The first would be to try using a counterfactual algorithm that could generate synthetic experiences from the reward machine. The synthetic experiences could help the agent in learning faster as they should decrease the number of samples needed to learn an adequate policy. The second would be to limit the set of plans that are used in the synthesis of the reward machine. In practice, different partial-order plans might be redundant and it might not be necessary to include all of them. This could be achieved by using techniques from planning, such as top- k planning (Katz and Lee, 2023).

8.1.2 Chapter 5: Synthesis of Reward Machines in Cooperative Multi-Agent Reinforcement Learning

After synthesising reward machines in single-agent RL, in Chapter 5 we have seen how reward machines can be synthesised in cooperative multi-agent RL. Motivated by the work of Neary et al. (2021) in MARL and the literature on synthesis of reward machines in the single-agent case, we aimed at providing a procedure that could be used to automatically synthesise reward machines in MARL.

In the approach presented in this chapter we propose to specify tasks for cooperating coalitions of agents via formulas of Alternating-time Temporal Logic (ATL). Given an ATL formula as input, we use a model checker for ATL and produce a strategy for the coalition (if there is any) which can be used to synthesise the coalition's reward machine. We have employed the approach proposed in (Neary et al., 2021) to train agents with RMs in a multi-agent scenario. Hence, from the coalition's reward machine we obtain a reward machine per each agent. We then train each agent individually with its RM, assuming that all the other agents in the coalition will eventually achieve their part of the task. Finally, we have compared our automatically synthesised reward machines to those of other works in the literature, including the aforementioned (Neary et al., 2021). We reported results showing how the performance of agents trained with our synthesised RMs is on the same level, and better in some cases, of that of agents trained with hand-defined RMs.

There are several research directions that can be explored to extend the work of this chapter. A first idea would be to drop the assumption that all agents are

cooperative. This would require an extension of the ATL fragment that we have used, as currently it only allows coalitional modalities where the quantified coalition is the grand coalition, i.e., the set of all agents. Another possibility would be to obtain all possible witnesses of the ATL formula, obtaining a reward machine similar to the maximally permissive reward machines from Chapter 4. However, for this direction the fact that the RMs represent multiple strategies would require the agents to coordinate on which strategy to follow. This coordination issue could be addressed, for example, by allowing the agents to communicate with each other.

8.1.3 Chapter 6: Defining Pushdown Reward Machines

In Chapter 6 we have presented the last work related to reward machines in this dissertation. In this chapter, we have introduced pushdown reward machines, an extension of reward machines obtained by equipping them with a stack. This addressed a clear research direction that has been present since the introduction of reward machines.

As pushdown reward machines are based on deterministic pushdown automata, they can encode reward functions which reward behaviours representable as deterministic context-free languages. As such, pushdown reward machines are strictly more expressive than reward machines, since the latter can only encode reward functions that reward behaviours representable as regular languages, a strict subset of deterministic context-free languages. We have defined two kinds of policies that can be learned when using pushdown reward machines: one in which the agent has access to the entire contents of the stack, and one where it has access only to the top k symbols (for a fixed constant k) of the stack. We have compared theoretically pushdown reward machines with counting reward automata (Bester et al., 2024), another extension of reward machines, by providing expressivity and space complexity (with respect to the increase in the policy’s state space) results. Then, we have proposed an extension of the counterfactual algorithm for reward machines specifically made for pushdown reward machines. Finally, we have presented experimental results. We have seen how in practice pushdown reward machines can, in some cases, lead to more efficient training than counting reward automata. We have also considered a domain with continuous state and action spaces, showing how pushdown reward machines can also be used in such cases.

As we have outlined in the chapter, possible future directions from this work include synthesis of pushdown reward machines using $LR(k)$ grammars, which are known to generate deterministic context-free languages. In the context of partially-observable RL, the stack of pushdown reward machines could also be used as a way of providing the agent with extra memory.

8.1.4 Chapter 7: Enforcing Safety in Reinforcement Learning with Pure-Past LTL

Finally, in Chapter 7, we have shifted our attention to safe reinforcement learning. The underlying motivation to this work was the need for approaches which could *provably* guarantee the satisfaction of temporal safety constraints at all times. In other

words, we require our approach to ensure that the agent will never take an action that would lead to a safety violation, not even during training. We achieved this by constraining action selection for the agent: intuitively, if an action would lead to a constraint violation, the agent is simply not allowed to take such action.

To constrain the agent from selecting actions that would violate the temporal safety constraints, we have employed Pure-Past Linear-time Temporal Logic (PPLTL). Each action is paired with a PPLTL formula which is evaluated at each timestep: the agent is allowed to take an action if and only if the action's formula is currently true. As the agent is allowed to take an action only if its corresponding PPLTL formula is true, our proposed approach provably ensures satisfaction of the safety constraints, both during and after training. We have related our approach to shields (Alshiekh et al., 2018), showing how both approaches can encode the same set of constraints. Finally, we also evaluated our approach in several domains, showing how in practice it ensures the satisfactions of safety constraints.

Future directions for this work include an extension of the approach to cases in which the action space is not finite. A possible way to tackle this would be to discretize the action space, thus re-obtaining a finite action space. Further, adapting the approaches proposed in (Stolz et al., 2024) to incorporate PPLTL constraints could also be an interesting way to approach this direction. Another interesting future direction would be to extend the work to the multi-agent case, as was done with shields in (ElSayed-Aly et al., 2021). Incorporating safety constraints via PPLTL in a multi-agent case could be done in several ways: for example, by constraining either joint or individual actions, or by having different constraints for each agent.

8.2 Future Research

We now outline possible future research directions in various subfields of reinforcement learning. For future directions related to the works presented in this thesis, we refer the reader to the previous section and to the concluding sections from Chapter 4, Chapter 5, Chapter 6, and Chapter 7.

8.2.1 Practical Applications

All the work in this dissertation has been conducted in simulated abstract environments. As AI should eventually be deployed in society, a first step to take would be to consider applications of our approaches to other, more practical, research fields. We envision several possible applications for the approaches proposed in this thesis in real-life domains.

Reward machines have already been employed in works in robotics, see, e.g., (Camacho, Varley et al., 2021) and (Baert et al., 2024). The developments in this dissertation on reward machines could thus be used also in the context of robotics. For example, applications of RMs to robotics have so far been confined to the single-agent case; by leveraging the approach proposed in Chapter 5, it would be possible to extend this application also to the multi-agent case and to allow the synthesis of reward machines from formal specifications for robotics tasks.

Reinforcement learning has also seen extensive application in domains related to healthcare (Yu et al., 2020). One of the areas in which RL is mostly applied in healthcare is that of *dynamic treatment regimes* (DTRs), where the main objective is to provide courses of actions to treat patients with long-lasting diseases. In this context, we believe that several ideas and approaches from this dissertation could be useful. For example, the temporal aspects of these treatments could be easily encoded and used via reward machines and/or temporal logics. Concerning the latter, PPLTL naturally lends itself to this safety-critical scenario, given that clinical decisions are made based on present and past data.¹

8.2.2 Partial Observability

Partial observability occurs naturally in several RL domains. For example, the agent might be a robot with a set of sensors that cannot perceive all the information needed to correctly model the environment's dynamics, or part of the dynamics are simply not known and as such cannot be encoded in the MDP. We have always assumed full observability in this dissertation; hence, extending the proposed approaches to a partially-observable setting is a natural research direction.

Reward machines have been used in partially-observable environments (see Section 3.1.1 for a short discussion about this), however, this direction has been explored exclusively in the single-agent case. As such, using RMs in a partially-observable MARL setting is still an open direction.

Concerning safety in the context of POMDPs, (Carr et al., 2023) is the only work that enforces the satisfaction of safety constraints. This is done via shielding, an approach that we have compared to the PPAMs proposed in this dissertation. Although in the context of partially-observable planning, Sheng et al. (2024) proposes different ways of using shields in POMDPs, differentiating between *centralized* and *factored* shielding similarly to what is done in multi-agent RL in (ElSayed-Aly et al., 2021). The factored shielding in this case consists in having multiple shields, each operating only on a specific portion of the POMDP. We believe that a similar approach could be taken also with PPAMs, and could be an interesting research direction.

8.2.3 Multi-Objective Reinforcement Learning

The final direction we propose lies in *multi-objective* reinforcement learning (MORL) (C. F. Hayes et al., 2022). MORL extends the reinforcement learning framework by presenting the agent with multiple goals instead of only one. This is done by having, instead of a scalar reward signal, a *vectorial* reward signal, i.e., the agent receives a tuple of rewards (one per each objective) at each timestep instead of a single one. As MORL can be used to train agent to take into account more objectives, and thus goals from different stakeholders, it easily lends itself to real-life domains. A peculiarity of MORL is that policy optimality depends on how the vectorial reward signal is aggregated. For example, a policy π_1 might maximize the return on an objective o_1 , whereas another policy π_2 might maximize the return on objective o_2 . Both policies

¹From (Yu et al., 2020): “A DTR is composed of a sequence of decision rules to determine the course of actions [...] at a time point according to the current health status and prior treatment history of an individual patient”.

are equally “optimal” in the sense that, so long as we do not know which objective the agent should maximize, they do not dominate each other. Aggregating the vectorial reward is done via a *utility function*, which is often assumed to be unknown a-priori. There are several reasons motivating the choice of leaving the utility function unknown in MORL, e.g., the fact that fixing it can collapse a MORL instance to a single-objective RL one in the case of linear utility functions, or simply because it is not known a-priori which objectives should be prioritized. Further, as MORL is more user-centric, the utility function is left unknown to account for the fact that different users have different utilities.² As policy optimality depends ultimately on the utility function, usually in MORL it is not possible to directly obtain optimal policies. Hence, MORL learning algorithms do not produce a single policy in output, but instead *solution sets*.

The research directions that have been explored in this dissertation could be extended to MORL. In (Aronis, 2025), reward machines have been extended to the multi-objective case by giving in output vectorial reward signals instead of scalar ones. For future efforts, it would be interesting to propose approaches that automatically synthesise reward machines for MORL using declarative AI techniques, as we have done in this dissertation. Compared to single-objective RL, the literature on *safe* MORL is still not as developed. Hence, proposing something similar to shields or our pure-past action masks (see Chapter 7) could be a way to address this problem. Following, we cite some works that have tried to develop safe MORL techniques. Sandy Huang et al. (2022) try to address this issue by extending the standard multi-objective MDP used in MORL to include cost functions. Policies are then considered safe if they do not exceed given cost thresholds for all the cost functions. Gu, Sel et al. (2025) also try to tackle this direction, and propose an approach which uses cost functions and which has guarantees of converging to policies that do not exceed the cost thresholds. While these approaches could be used in settings where a certain degree of safety violation is allowed, they are not suitable for safety-critical domains where agents should always satisfy the safety constraints. We believe that proposing provably safe approaches for MORL could be an interesting research gap to try to address by leveraging pure-past action masks and shields.

8.3 Conclusion

We have explored applications of approaches and techniques from declarative AI to reinforcement learning. A large part of this dissertation was devoted to reward machines. We have provided new ways of automatically synthesising reward machines, and have extended them to make them more expressive. We hope that these advancements will ease possible applications of reward machines in other areas of RL and research fields. We have proposed then an approach that can ensure the satisfaction of safety constraints by employing temporal logic. Finally, we hope that the research presented in this dissertation will inspire future work in applying techniques from declarative AI to reinforcement learning.

²For a more detailed discussion about aspects concerning utility functions, we refer the reader to Sections 1 and 5.1 of (C. F. Hayes et al., 2022).



Bibliography

- [1] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall Englewood Cliffs, 1973.
- [2] Natasha Alechina, Nils Bulling, Mehdi Dastani and Brian Logan. ‘Practical Run-Time Norm Enforcement with Bounded Lookahead’. In: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2015, pp. 443–451.
- [3] Natasha Alechina, Mehdi Dastani and Brian Logan. ‘Norm approximation for imperfect monitors’. In: *Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2014, pp. 117–124.
- [4] Shayan M. Alsadat, Jean-Raphaël Gaglione, Daniel Neider, Ufuk Topcu and Zhe Xu. ‘Using large language models to automate and expedite reinforcement learning with reward machine’. In: *Proceedings of the 2025 American Control Conference (ACC)*. 2025, pp. 206–211.
- [5] Shayan M. Alsadat and Zhe Xu. ‘Multi-Agent Reinforcement Learning in Non-Cooperative Stochastic Games Using Large Language Models’. In: *IEEE Control Systems Letters* 8 (2024), pp. 2757–2762.
- [6] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum and Ufuk Topcu. ‘Safe Reinforcement Learning via Shielding’. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*. 2018, pp. 2669–2678.
- [7] Rajeev Alur, Thomas A. Henzinger and Orna Kupferman. ‘Alternating-Time Temporal Logic’. In: *Journal of the ACM* 49.5 (2002), pp. 672–713.
- [8] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman and Dan Mané. *Concrete Problems in AI Safety*. arXiv preprint arXiv:1606.06565. 2016.
- [9] Davide Ancona, Luca Franceschini, Angelo Ferrando and Viviana Mascardi. ‘RML: Theory and practice of a domain specific language for runtime verification’. In: *Science of Computer Programming* 205 (2021), p. 102610.

- [10] Jacob Andreas, Dan Klein and Sergey Levine. ‘Modular Multitask Reinforcement Learning with Policy Sketches’. In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. 2017, pp. 166–175.
- [11] Leo Ardon, Daniel Furelos-Blanco, Roko Parac and Alessandra Russo. ‘FORM: Learning Expressive and Transferable First-Order Logic Reward Machines’. In: *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2025, pp. 142–151.
- [12] Leo Ardon, Daniel Furelos-Blanco and Alessandra Russo. ‘Learning Reward Machines in Cooperative Multi-agent Tasks’. In: *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2023, pp. 43–59.
- [13] Panos Aronis. ‘Leveraging Reward Machines for Efficient Multi-Objective Reinforcement Learning’. MA thesis. Utrecht University (The Netherlands), 2025.
- [14] Fahiem Bacchus, Craig Boutilier and Adam J. Grove. ‘Rewarding Behaviors’. In: *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*. 1996.
- [15] Mattijs Baert, Sam Leroux and Pieter Simoens. *Reward Machine Inference for Robotic Manipulation*. arXiv preprint arXiv:2412.10096. 2024.
- [16] Arko Banerjee, Kia Rahmani, Joydeep Biswas and Isil Dillig. ‘Dynamic model predictive shielding for provably safe reinforcement learning’. In: *Proceedings of the 38th Conference Advances in Neural Information Processing Systems (NeurIPS)*. 2024, pp. 100131–100159.
- [17] Andrew Barto and Michael Duff. ‘Monte Carlo matrix inversion and reinforcement learning’. In: *Proceedings of the 7th Conference on Advances in Neural Information Processing Systems (NIPS)*. 1994, pp. 687–694.
- [18] Richard Bellman. ‘Dynamic programming and stochastic control processes’. In: *Information and control* 1.3 (1958), pp. 228–239.
- [19] Richard Bellman. ‘Dynamic Programming’. In: *Science* 153.3731 (1966), pp. 34–37.
- [20] Yoshua Bengio, Jérôme Louradour, Ronan Collobert and Jason Weston. ‘Curriculum learning’. In: *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*. 2009, pp. 41–48.
- [21] Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig and Andreas Krause. ‘Safe model-based reinforcement learning with stability guarantees’. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*. 2017, pp. 908–919.
- [22] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Mi-

- chael Petrov, Henrique P. de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski and Susan Zhang. *Dota 2 with Large Scale Deep Reinforcement Learning*. arXiv preprint arXiv:1912.06680. 2019.
- [23] Tristan Bester, Benjamin Rosman, Steven James and Geraud Nangue Tasse. *Counting Reward Automata: Sample Efficient Reinforcement Learning Through the Exploitation of Reward Function Structure*. arXiv preprint arXiv:2312.11364. 2024.
 - [24] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli and Yaniv Sa'ar. 'Synthesis of Reactive(1) designs'. In: *Journal of Computer and System Sciences* 78.3 (2012), pp. 911–938.
 - [25] Roderick Bloem, Bettina Könighofer, Robert Könighofer and Chao Wang. 'Shield synthesis: Runtime enforcement for reactive systems'. In: *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2015, pp. 533–548.
 - [26] Roderick Bloem, Sven Schewe and Ayrat Khalimov. 'CTL* synthesis via LTL synthesis'. In: *Electronic Proceedings in Theoretical Computer Science* 260 (2017), pp. 4–22.
 - [27] Luigi Bonassi, Giuseppe De Giacomo, Marco Favorito, Francesco Fuggitti, Alfonso Emilio Gerevini and Enrico Scala. 'Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic'. In: *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*. 2023, pp. 61–69.
 - [28] Craig Boutilier. 'Planning, Learning and Coordination in Multiagent Decision Processes'. In: *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*. 1996, pp. 195–210.
 - [29] Ronen I. Brafman and Giuseppe De Giacomo. 'Regular Decision Processes: A Model for Non-Markovian Domains'. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence, (IJCAI)*. 2019, pp. 5516–5522.
 - [30] Ronen I. Brafman and Giuseppe De Giacomo. 'Regular decision processes'. In: *Artificial Intelligence* 331 (2024), p. 104113.
 - [31] Asger H. Brorholt, Kim G. Larsen and Christian Schilling. *Compositional Shielding and Reinforcement Learning for Multi-Agent Systems*. arXiv preprint arXiv:2410.10460. 2025.
 - [32] Alberto Camacho, Rodrigo A. Toro Icarte, Toryn Q. Klassen, Richard Valenzano and Sheila A. McIlraith. 'LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning'. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*. 2019, pp. 6065–6073.
 - [33] Alberto Camacho, Jacob Varley, Andy Zeng, Deepali Jain, Atil Iscen and Dmitry Kalashnikov. 'Reward Machines for Vision-Based Robotic Manipula-

- tion'. In: *Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 14284–14290.
- [34] Steven Carr, Nils Jansen, Sebastian Junges and Ufuk Topcu. 'Safe reinforcement learning via shielding under partial observability'. In: *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*. 2023, pp. 14748–14756.
- [35] Roger C. Castanyer, Faisal Mohamed, Pablo S. Castro, Cyrus Neary and Glen Berseth. *ARM-FM: Automated Reward Machines via Foundation Models for Compositional Reinforcement Learning*. arXiv preprint arXiv:2510.14176. 2025.
- [36] David Chapman. 'Planning for Conjunctive Goals'. In: *Artificial Intelligence* 32.3 (1987), pp. 333–377.
- [37] Richard Cheng, Gábor Orosz, Richard M. Murray and Joel W. Burdick. 'End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks'. In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 3387–3395.
- [38] Luca Chittaro and Angelo Montanari. 'Temporal representation and reasoning in artificial intelligence: Issues and approaches'. In: *Annals of Mathematics and Artificial Intelligence* 28.1 (2000), pp. 47–106.
- [39] Noam Chomsky and Marcel P. Schützenberger. 'The algebraic theory of context-free languages'. In: *Studies in Logic and the Foundations of Mathematics*. Ed. by Paul Braffort and David Hirschberg. Vol. 26. Elsevier, 1959, pp. 118–161.
- [40] Edmund M. Clarke and E. Allen Emerson. 'Design and synthesis of synchronization skeletons using branching time temporal logic'. In: *Proceedings of the Workshop on Logics of Programs*. 1981, pp. 52–71.
- [41] Mika Cohen, Mads Dam, Alessio Lomuscio and Francesco Russo. 'Abstraction in model checking multi-agent systems'. In: *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009, pp. 945–952.
- [42] Jan Corazza, Ivan Gavran and Daniel Neider. 'Reinforcement Learning with Stochastic Reward Machines'. In: *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*. 2022, pp. 6429–6436.
- [43] Kai Cui, Anam Tahir, Gizem Ekinici, Ahmed Elshamanhory, Yannick Eich, Mengguang Li and Heinz Koepl. *A Survey on Large-Population Systems and Scalable Multi-Agent Reinforcement Learning*. arXiv preprint arXiv:2209.03859. 2022.
- [44] Emanuele De Angelis, Alberto Pettorossi and Maurizio Proietti. 'Synthesizing concurrent programs using answer set programming'. In: *Fundamenta Informaticae* 120.3–4 (2012), pp. 205–229.
- [45] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti and Sasha Rubin. 'Pure-Past Linear Temporal and Dynamic Logic on Finite Traces'. In: *Proceed-*

- ings of the 29th International Joint Conference on Artificial Intelligence (IJCAI). 2020, pp. 4959–4965.
- [46] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito and Fabio Patrizi. ‘Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications’. In: *Proceedings of the 29th International Conference on Automated Planning and Scheduling*. 2019, pp. 128–136.
 - [47] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito and Fabio Patrizi. ‘Restraining Bolts for Reinforcement Learning Agents’. In: *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*. 2020, pp. 13659–13662.
 - [48] Giuseppe De Giacomo and Moshe Y. Vardi. ‘Linear temporal logic and linear dynamic logic on finite traces’. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*. 2013, pp. 854–860.
 - [49] Giuseppe De Giacomo and Moshe Y. Vardi. ‘Synthesis for LTL and LDL on finite traces’. In: *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*. 2015, pp. 1558–1564.
 - [50] Thomas Dietterich. ‘Hierarchical reinforcement learning with the MAXQ value function decomposition’. In: *Journal of Artificial Intelligence Research* 13.1 (2000), pp. 227–303.
 - [51] Daniel Donnelly, Angelo Ferrando and Francesco Belardinelli. *Expressive Reward Synthesis with the Runtime Monitoring Language*. arXiv preprint arXiv:2510.16185. 2025.
 - [52] Sašo Džeroski, Luc De Raedt and Hendrik Blockeel. ‘Relational reinforcement learning’. In: *Proceedings of the 8th Conference on Inductive Logic Programming*. 1998, pp. 11–22.
 - [53] Sašo Džeroski, Luc De Raedt and Kurt Driessens. ‘Relational reinforcement learning’. In: *Machine learning* 43.1 (2001), pp. 7–52.
 - [54] Joe Eappen and Suresh Jagannathan. ‘DistSPECTRL: Distributing Specifications in Multi-Agent Reinforcement Learning Systems’. In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*. 2023, pp. 233–250.
 - [55] Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster and Shimon Whiteson. ‘Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning’. In: *Proceedings of the 37th Conference Advances in Neural Information Processing Systems (NeurIPS)*. 2023, pp. 37567–37593.
 - [56] Ingy ElSayed-Aly, Suda Bharadwaj, Christopher Amato, Rüdiger Ehlers, Ufuk Topcu and Lu Feng. ‘Safe Multi-Agent Reinforcement Learning via Shielding’. In: *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (IJCAI)*. 2021, pp. 483–491.

- [57] Amin Falah, Shibashis Guha and Ashutosh Trivedi. ‘Continuous-Time Reward Machines’. In: *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI)*. 2025, pp. 5056–5064.
- [58] Eugene A. Feinberg. ‘Total expected discounted reward MDPs: Existence of optimal policies’. In: *Wiley Encyclopedia of Operations Research and Management Science*. Ed. by James J. Cochran. John Wiley & Sons, Hoboken, NJ, 2011.
- [59] Angelo Ferrando and Vadim Malvone. ‘VITAMIN: A compositional framework for model checking of multi-agent systems’. In: *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART)*. 2025, pp. 648–655.
- [60] Patrick C. Fischer. ‘Turing machines with restricted memory access’. In: *Information and Control* 9.4 (1966), pp. 364–379.
- [61] Patrick C. Fischer, Albert R. Meyer and Arnold L. Rosenberg. ‘Counter machines and counter languages’. In: *Mathematical Systems Theory* 2.3 (1968), pp. 265–283.
- [62] Jie Fu and Ufuk Topcu. ‘Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints’. In: *Proceedings of the 10th Robotics: Science and Systems Conference (RSS)*. 2014.
- [63] Scott Fujimoto, Herke Hoof and David Meger. ‘Addressing function approximation error in actor-critic methods’. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018, pp. 1587–1596.
- [64] Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda and Alessandra Russo. ‘Hierarchies of Reward Machines’. In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. 2023, pp. 10494–10541.
- [65] Alfredo Gabaldon. ‘Non-Markovian control in the Situation Calculus’. In: *Artificial Intelligence* 175.1 (2011), pp. 25–48.
- [66] Peter Gammie and Ron van der Meyden. ‘MCK: Model Checking the Logic of Knowledge’. In: *Computer Aided Verification*. 2004, pp. 479–483.
- [67] Maor Gaon and Ronen I. Brafman. ‘Reinforcement Learning with Non-Markovian Rewards’. In: *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*. 2020, pp. 3980–3987.
- [68] Javier García and Fernando Fernández. ‘A Comprehensive Survey on Safe Reinforcement Learning’. In: *Journal of Machine Learning Research* 16.42 (2015), pp. 1437–1480.
- [69] Shangding Gu, Bilgehan Sel, Yuhao Ding, Lu Wang, Qingwei Lin, Alois Knoll and Ming Jin. ‘Safe and Balanced: A Framework for Constrained Multi-Objective Reinforcement Learning’. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 47.5 (2025), pp. 3322–3331.

- [70] Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang and Alois Knoll. *A Review of Safe Reinforcement Learning: Methods, Theory and Applications*. arXiv preprint arXiv:2205.10330. 2024.
- [71] Xianping Guo and Onésimo Hernández-Lerma. ‘Continuous-Time Markov Decision Processes’. In: *Continuous-Time Markov Decision Processes: Theory and Applications*. Springer Berlin Heidelberg, 2009, pp. 9–18.
- [72] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel and Sergey Levine. ‘Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor’. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018, pp. 1861–1870.
- [73] Dylan Hadfield-Menell, Anca D. Dragan, Pieter Abbeel and Stuart Russell. ‘The Off-Switch Game’. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence, (IJCAI)*. 2017, pp. 220–227.
- [74] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi and Dominik Wojtczak. ‘Omega-Regular Objectives in Model-Free Reinforcement Learning’. In: *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2019, pp. 395–412.
- [75] Shuai Han, Mehdi Dastani and Shihan Wang. ‘Learning Reward Structure with Subtasks in Reinforcement Learning’. In: *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI)*. 2024, pp. 2282–2289.
- [76] Hans Hansson and Bengt Jonsson. ‘A logic for reasoning about time and reliability’. In: *Formal aspects of computing* 6.5 (1994), pp. 512–535.
- [77] Hosein Hasanbeig, Natasha Y. Jeppu, Alessandro Abate, Tom Melham and Daniel Kroening. ‘Symbolic Task Inference in Deep Reinforcement Learning’. In: *Journal of Artificial Intelligence Research* 80 (2024), pp. 1099–1137.
- [78] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham and Daniel Kroening. ‘Deepsynth: Automata synthesis for automatic task segmentation in deep reinforcement learning’. In: *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*. 2021, pp. 7647–7656.
- [79] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J. Pappas and Insup Lee. ‘Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees’. In: *Proceedings of the 58th Conference on Decision and Control (CDC)*. 2019, pp. 5338–5343.
- [80] Mohammadhosein Hasanbeig, Daniel Kroening and Alessandro Abate. ‘Deep Reinforcement Learning with Temporal Logics’. In: *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 2020, pp. 1–22.

- [81] Mohammadhosein Hasanbeig, Daniel Kroening and Alessandro Abate. ‘LCRL: Certified Policy Synthesis via Logically-Constrained Reinforcement Learning’. In: *Proceedings of the 19th International Conference on Quantitative Evaluation of Systems (QEST)*. 2022, pp. 217–231.
- [82] Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew and Diederik M. Roijers. ‘A practical guide to multi-objective reinforcement learning and planning’. In: *Autonomous Agents and Multi-Agent Systems* 36.1 (2022).
- [83] Floris den Hengst, Vincent Francois-Lavet, Mark Hoogendoorn and Frank van Harmelen. ‘Reinforcement Learning with Option Machines’. In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI*. 2022, pp. 2909–2915.
- [84] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag and Enrique M. de Cote. *A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity*. arXiv preprint arXiv:1707.09183. 2019.
- [85] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar and David Silver. ‘Rainbow: Combining Improvements in Deep Reinforcement Learning’. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*. 2018, pp. 3215–3222.
- [86] Sepp Hochreiter and Jürgen Schmidhuber. ‘Long Short-Term Memory’. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.
- [87] Jueming Hu, Zhe Xu, Weichang Wang, Guannan Qu, Yutian Pang and Yongming Liu. ‘Decentralized Graph-Based Multi-Agent Reinforcement Learning Using Reward Machines’. In: *Neurocomputing* 564.C (2024), p. 126974.
- [88] Sandy Huang, Abbas Abdolmaleki, Giulia Vezzani, Philemon Brakel, Daniel J. Mankowitz, Michael Neunert, Steven Bohez, Yuval Tassa, Nicolas Heess, Martin Riedmiller and Raia Hadsell. ‘A Constrained Multi-Objective Reinforcement Learning Framework’. In: *Proceedings of the 5th Conference on Robot Learning (CoRL)*. 2022, pp. 883–893.
- [89] Shengyi Huang and Santiago Ontañón. ‘A Closer Look at Invalid Action Masking in Policy Gradient Algorithms’. In: *Proceedings of the 35th International FLAIRS Conference (FLAIRS)*. 2022.
- [90] Nathan Hunt, Nathan Fulton, Sara Magliacane, Trong Nghia Hoang, Subhro Das and Armando Solar-Lezama. ‘Verifiably Safe Exploration for End-to-End Reinforcement Learning’. In: *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control (HSCC)*. 2021, pp. 1–11.

- [91] León Illanes, Xi Yan, Rodrigo A. Toro Icarte and Sheila A. McIlraith. ‘Symbolic Planning and Model-Free Reinforcement Learning: Training Taskable Agents’. In: *Proceedings of the 4th Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*. 2019, pp. 191–195.
- [92] León Illanes, Xi Yan, Rodrigo A. Toro Icarte and Sheila A. McIlraith. ‘Symbolic Plans as High-Level Instructions for Reinforcement Learning’. In: *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*. 2020, pp. 540–550.
- [93] Yogi Joshi, Guy M. Tchamgoue and Sebastian Fischmeister. ‘Runtime verification of LTL on lossy traces’. In: *Proceedings of the Symposium on Applied Computing (SAC)*. 2017, pp. 1379–1386.
- [94] Kishor Jothimurugan, Rajeev Alur and Osbert Bastani. ‘A Composable Specification Language for Reinforcement Learning Tasks’. In: *Advances in Neural Information Processing Systems 32 (NeurIPS)*. 2019.
- [95] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani and Rajeev Alur. ‘Compositional reinforcement learning from logical specifications’. In: *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS)*. 2021, pp. 10026–10039.
- [96] Sham Kakade and John Langford. ‘Approximately optimal approximate reinforcement learning’. In: *Proceedings of the 19th International Conference on Machine Learning (ICML)*. 2002, pp. 267–274.
- [97] Malvin H. Kalos and Paula A. Whitlock. *Monte carlo methods*. John Wiley & Sons, 2009.
- [98] Anssi Kanervisto, Christian Scheller and Ville Hautamäki. ‘Action space shaping in deep reinforcement learning’. In: *Proceedings of the 2020 IEEE Conference on Games (CoG)*. 2020, pp. 479–486.
- [99] Andrej Karpathy. *REINFORCEjs: WaterWorld demo*. <https://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html>. 2015.
- [100] Michael Katz and Junkyu Lee. ‘ K_* Search over Orbit Space for Top-k Planning’. In: *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI)*. 2023, pp. 5368–5376.
- [101] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley and Amar Shah. ‘Learning to Drive in a Day’. In: *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 8248–8254.
- [102] Ayrat Khalimov. ‘Reactive Synthesis: Branching Logics and Parameterized Systems’. PhD thesis. Graz University of Technology (Austria), 2018.
- [103] Ayrat Khalimov and Roderick Bloem. ‘Bounded synthesis for streett, rabin, and CTL*’. In: *International Conference on Computer Aided Verification (CAV)*. Springer. 2017, pp. 333–352.

- [104] Tobias Klenze, Sam Bayless and Alan J Hu. ‘Fast, flexible, and minimal CTL synthesis via SMT’. In: *Proceedings of the 28th International Conference on Computer Aided Verification (CAV)*. 2016, pp. 136–156.
- [105] Donald E. Knuth. ‘On the translation of languages from left to right’. In: *Information and Control* 8.6 (1965), pp. 607–639.
- [106] Harsha Kokel, Arjun Manoharan, Sriraam Natarajan, Balaraman Ravindran and Prasad Tadepalli. ‘RePReL: Integrating Relational Planning and Reinforcement Learning for Effective Abstraction’. In: *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS)*. 2021, pp. 533–541.
- [107] John Komp, Dananjay Srinivas, Maria Pacheco and Ashutosh Trivedi. *Show, Don’t Tell: Learning Reward Machines from Demonstrations for Reinforcement Learning-Based Cardiac Pacemaker Synthesis*. arXiv preprint arXiv:2411.01750. 2024.
- [108] Bettina Könighofer, Roderick Bloem, Nils Jansen, Sebastian Junges and Stefan Pranger. ‘Shields for Safe Reinforcement Learning’. In: *Communications of the ACM* 68.11 (2025), pp. 80–90.
- [109] Cevahir Koprulu and Ufuk Topcu. ‘Reward-machine-guided, self-paced reinforcement learning’. In: *Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence (UAI)*. 2023, pp. 1121–1131.
- [110] Hanna Krasowski, Jakob Thumm, Marlon Müller, Lukas Schäfer, Xiao Wang and Matthias Althoff. ‘Provably Safe Reinforcement Learning: Conceptual Analysis, Survey, and Benchmarking’. In: *Transactions on Machine Learning Research* (2023).
- [111] Hanna Krasowski, Xiao Wang and Matthias Althoff. ‘Safe reinforcement learning for autonomous lane changing using set-based prediction’. In: *Proceedings of the 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 2020, pp. 1–7.
- [112] Vipin Kumar and Laveen N. Kanal. ‘The CDP: A Unifying Formulation for Heuristic Search, Dynamic Programming, and Branch-and-Bound’. In: *Search in Artificial Intelligence*. Ed. by Laveen Kanal and Vipin Kumar. Springer New York, 1988, pp. 1–27.
- [113] Orna Kupferman and Moshe Y. Vardi. ‘Model Checking of Safety Properties’. In: *Formal Methods in System Design* 19.3 (2001), pp. 291–314.
- [114] Damian Kurpiewski, Mateusz Kaminski and Wojciech Jamroga. ‘STV+FLY: On-the-Fly Model Checking of Strategic Ability in Multi-Agent Systems’. In: *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI)*. 2024, pp. 4483–4486.
- [115] Ankita Kushwaha, Kiran Ravish, Preeti Lamba and Pawan Kumar. *A Survey of Safe Reinforcement Learning and Constrained MDPs: A Technical Survey on Single-Agent and Multi-Agent Safety*. arXiv preprint arXiv:2505.17342. 2025.

- [116] Romain Laroche, Paul Trichelair and Remi Tachet Des Combes. ‘Safe policy improvement with baseline bootstrapping’. In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019, pp. 3652–3661.
- [117] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau and Shane Legg. *AI Safety Gridworlds*. arXiv preprint arXiv:1711.09883. 2017.
- [118] Kristina Levina, Nikolaos Pappas, Athanasios Karapantelakis, Aneta V. Feljan and Jendrik Seipp. ‘Numeric Reward Machines’. In: *Proceedings of the ICAPS 2024 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*. 2024.
- [119] Andrew C. Li, Zizhao Chen, Toryn Q. Klassen, Pashootan Vaezipoor, Rodrigo Toro A. Icarte and Sheila A. McIlraith. ‘Reward Machines for Deep RL in Noisy and Uncertain Environments’. In: *Advances in Neural Information Processing Systems*. 2024, pp. 110341–110368.
- [120] Xiao Li, Zachary Serlin, Guang Yang and Calin Belta. ‘A formal methods approach to interpretable reinforcement learning for robotic planning’. In: *Science Robotics* 4.37 (2019), eaay6276.
- [121] Xiao Li, Cristian-Ioan Vasile and Calin Belta. ‘Reinforcement learning with temporal logic rewards’. In: *Proceedings of the 30th International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 3834–3839.
- [122] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan and Ion Stoica. ‘RLlib: Abstractions for Distributed Reinforcement Learning’. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018, pp. 3053–3062.
- [123] Orna Lichtenstein, Amir Pnueli and Lenore Zuck. ‘The glory of the past’. In: *Proceedings of the Conference on Logics of Programs*. 1985, pp. 196–218.
- [124] Michael L. Littman. ‘Markov Games as a Framework for Multi-Agent Reinforcement Learning’. In: *Proceedings of the 11th International Conference on Machine Learning (ICML)*. 1994, pp. 157–163.
- [125] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen and James MacGlashan. *Environment-independent task specifications via GLTL*. arXiv preprint arXiv:1704.04341. 2017.
- [126] Jung-Chun Liu, Chi-Hsien Chang, Shao-Hua Sun and Tian-Li Yu. ‘Integrating planning and deep reinforcement learning via automatic induction of task substructures’. In: *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. 2024.
- [127] Keliang Liu, Dingkan Yang, Ziyun Qian, Weijie Yin, Yuchi Wang, Hongsheng Li, Jun Liu, Peng Zhai, Yang Liu and Lihua Zhang. *Reinforcement Learning Meets Large Language Models: A Survey of Advancements and Applications Across the LLM Lifecycle*. arXiv preprint arXiv:2509.16679. 2025.

- [128] Alessio Lomuscio, Hongyang Qu and Franco Raimondi. 'MCMAS: An Open-Source Model Checker for the Verification of Multi-Agent Systems'. In: *International Journal on Software Tools for Technology Transfer* 19.1 (2017), pp. 9–30.
- [129] Alessio Lomuscio and Franco Raimondi. 'Model checking knowledge, strategies, and games in multi-agent systems'. In: *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2006, pp. 161–168.
- [130] Michael Luttenberger, Philipp J. Meyer and Salomon Sickert. 'Practical synthesis of reactive systems from LTL specifications via parity games'. In: *Acta Informatica* 57 (2020), pp. 3–36.
- [131] Daoming Lyu, Fangkai Yang, Bo Liu and Steven Gustafson. 'SDRL: Interpretable and Data-Efficient Deep Reinforcement Learning Leveraging Symbolic Planning'. In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 2970–2977.
- [132] Zohar Manna and Amir Pnueli. 'A Hierarchy of Temporal Properties'. In: *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC 1990)*. 1990, pp. 377–410.
- [133] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety*. Springer, 1995.
- [134] Gary Marcus. *Deep learning: A critical appraisal*. arXiv preprint arXiv:1801.00631. 2018.
- [135] George Rupert Mason, Radu Constantin Calinescu, Daniel Kudenko and Alec Banks. 'Assured reinforcement learning with formally verified abstract policies'. In: *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART)*. 2017, pp. 105–117.
- [136] René Mazala. 'Infinite Games'. In: *Automata Logics, and Infinite Games: A Guide to Current Research*. Ed. by Erich Grädel, Wolfgang Thomas and Thomas Wilke. Springer Berlin Heidelberg, 2002, pp. 23–38.
- [137] David McAllester and David Rosenblitt. 'Systematic Nonlinear Planning'. In: *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI)*. 1991, pp. 634–639.
- [138] John McCarthy and Patrick J. Hayes. 'Some philosophical problems from the standpoint of artificial intelligence'. In: *Readings in artificial intelligence*. Ed. by Bonnie L. Webber and Nils J. Nilsson. Elsevier, 1981, pp. 431–450.
- [139] Philipp J. Meyer, Salomon Sickert and Michael Luttenberger. 'Strix: Explicit Reactive Synthesis Strikes Back!' In: *Proceedings of the 30th International Conference on Computer Aided Verification (CAV)*. 2018, pp. 578–586.
- [140] Marvin L. Minsky. *Computation*. Prentice-Hall Englewood Cliffs, 1967.

- [141] Branka Mirchevska, Christian Pek, Moritz Werling, Matthias Althoff and Joschka Boedecker. ‘High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning’. In: *Proceedings of the 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 2156–2162.
- [142] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter and Jeff Dean. ‘A graph placement methodology for fast chip design’. In: *Nature* 594.7862 (2021), pp. 207–212.
- [143] Volodymyr Mnih, Adria P. Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver and Koray Kavukcuoglu. ‘Asynchronous methods for deep reinforcement learning’. In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. 2016, pp. 1928–1937.
- [144] Thomas M. Moerland, Joost Broekens, Aske Plaat and Catholijn M. Jonker. ‘Model-based reinforcement learning: A survey’. In: *Foundations and Trends® in Machine Learning* 16.1 (2023), pp. 1–118.
- [145] Cyrus Neary, Zhe Xu, Bo Wu and Ufuk Topcu. ‘Reward Machines for Co-operative Multi-Agent Reinforcement Learning’. In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AA-MAS)*. 2021, pp. 934–942.
- [146] XuanLong Nguyen and Subbarao Kambhampati. ‘Reviving Partial Order Planning’. In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*. 2001, pp. 459–466.
- [147] Haritz Odriozola-Olalde, Maider Zamalloa and Nestor Arana-Arexolaleiba. ‘Shielded Reinforcement Learning: A review of reactive methods for safe learning’. In: *Proceedings of the 2023 IEEE/SICE International Symposium on System Integration (SII)*. 2023, pp. 1–8.
- [148] Laurent Orseau and Stuart Armstrong. ‘Safely Interruptible Agents’. In: *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence (UAI)*. 2016, pp. 557–566.
- [149] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike and Ryan Lowe. *Training language models to follow instructions with human feedback*. arXiv preprint arXiv:2203.02155. 2022.
- [150] Liviu Panait and Sean Luke. ‘Cooperative Multi-Agent Learning: The State of the Art’. In: *Autonomous Agents and Multi-Agent Systems* 11.3 (2005), pp. 387–434.

- [151] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer and Stefano V. Albrecht. ‘Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks’. In: *Proceedings of the 35th Conference on Advances in Neural Information Processing Systems (NeurIPS)*. 2021.
- [152] Aske Plaat. *Deep Reinforcement Learning*. Springer Nature Singapore, 2022.
- [153] Amir Pnueli. ‘The temporal logic of programs’. In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS)*. 1977, pp. 46–57.
- [154] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.
- [155] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus and Noah Dormann. ‘Stable-Baselines3: Reliable reinforcement learning implementations’. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.
- [156] Adithya Ramesh and Balaraman Ravindran. *Physics-Informed Model-Based Reinforcement Learning*. arXiv preprint arXiv:2212.02179. 2023.
- [157] Gavin A. Rummery and Mahesan Niranjana. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, 1994.
- [158] Thomas Savage, Dongda Zhang, Max Mowbray and Ehecatl A. del Río-Chanona. ‘Model-free safe reinforcement learning for chemical processes using Gaussian processes’. In: *IFAC-PapersOnLine* 54.3 (2021), pp. 504–509.
- [159] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan and Philipp Moritz. ‘Trust Region Policy Optimization’. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. 2015, pp. 1889–1897.
- [160] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford and Oleg Klimov. *Proximal Policy Optimization Algorithms*. arXiv preprint arXiv:1707.06347. 2017.
- [161] Ankit Shah, Shen Li and Julie Shah. ‘Planning with uncertain specifications (puns)’. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3414–3421.
- [162] Shili Sheng, David Parker and Lu Feng. ‘Safe POMDP online planning via shielding’. In: *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 126–132.
- [163] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Thore Kavukcuoglu, Koray Graepel and Demis Hassabis. ‘Mastering the game of Go with deep neural networks and tree search’. In: *Nature* 529.7587 (2016), pp. 484–489.

-
- [164] Tom Silver, Kelsey Allen, Josh Tenenbaum and Leslie Kaelbling. *Residual Policy Learning*. arXiv preprint arXiv:1812.06298. 2019.
- [165] Thiago D. Simão, Nils Jansen and Matthijs T. J. Spaan. ‘AlwaysSafe: Reinforcement Learning without Safety Constraint Violations during Training’. In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2021, pp. 1226–1235.
- [166] Satinder P. Singh. ‘Reinforcement learning with a hierarchy of abstract models’. In: *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*. 1992, pp. 202–207.
- [167] Satinder P. Singh. ‘Transfer of learning by composing solutions of elemental sequential tasks’. In: *Machine learning 8.3* (1992), pp. 323–339.
- [168] Joar Skalse, Nikolaus Howe, Dmitrii Krashenninikov and David Krueger. ‘Defining and Characterizing Reward Gaming’. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*. 2022, pp. 9460–9471.
- [169] Sophia Smith, Cyrus Neary and Ufuk Topcu. ‘Automatic Decomposition of Reward Machines for Decentralized Multiagent Reinforcement Learning’. In: *Proceedings of the 62nd IEEE Conference on Decision and Control (CDC)*. 2023, pp. 5423–5430.
- [170] Matthijs T. J. Spaan. ‘Partially observable Markov decision processes’. In: *Reinforcement learning: State-of-the-art*. Ed. by Marco A. Wiering and Martijn van Otterlo. Springer, 2012, pp. 387–414.
- [171] Roland Stolz, Hanna Krasowski, Jakob Thumm, Michael Eichelbeck, Philipp Gassert and Matthias Althoff. ‘Excluding the irrelevant: Focusing reinforcement learning through continuous action masking’. In: *Proceedings of the 38th Conference Advances in Neural Information Processing Systems (NeurIPS)*. 2024, pp. 95067–95094.
- [172] Haolin Sun and Yves Lespérance. ‘Exploiting Reward Machines with Deep Reinforcement Learning in Continuous Action Domains’. In: *Proceedings of the 20th European Conference on Multi-Agent Systems (EUMAS)*. 2023, pp. 328–344.
- [173] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An Introduction*. MIT press, 1998.
- [174] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.
- [175] Richard S. Sutton, David McAllester, Satinder Singh and Yishay Mansour. ‘Policy Gradient Methods for Reinforcement Learning with Function Approximation’. In: *Proceedings of the 12th Conference on Advances in Neural Information Processing Systems (NIPS)*. 1999, pp. 1057–1063.

- [176] Richard S. Sutton, Doina Precup and Satinder Singh. ‘Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning’. In: *Artificial Intelligence* 112.1 (1999), pp. 181–211.
- [177] Lucas M. Tabajara and Moshe Y. Vardi. ‘LTLf Synthesis under Partial Observability: From Theory to Practice’. In: *Electronic Proceedings in Theoretical Computer Science* 326 (2020), pp. 1–17.
- [178] Ming Tan. ‘Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents’. In: *Proceedings of the 10th International Conference on Machine Learning (ICML)*. 1993, pp. 330–337.
- [179] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Zhaopeng Meng, Changjie Fan and Li Wang. *Hierarchical Deep Multiagent Reinforcement Learning with Temporal Abstraction*. arXiv preprint arXiv:1809.09332. 2019.
- [180] J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh and Praveen Ravi. *PettingZoo: Gym for Multi-Agent Reinforcement Learning*. arXiv preprint arXiv:2009.14471. 2021.
- [181] Rodrigo A. Toro Icarte. ‘Reward Machines’. PhD thesis. University of Toronto (Canada), 2022.
- [182] Rodrigo A. Toro Icarte, Toryn Q. Klassen, Richard Valenzano, Margarita P. Castro, Ethan Waldie and Sheila A. McIlraith. ‘Learning reward machines: A study in partially observable reinforcement learning’. In: *Artificial Intelligence* 323 (2023), p. 103989.
- [183] Rodrigo A. Toro Icarte, Toryn Q. Klassen, Richard Valenzano and Sheila A. McIlraith. ‘Using reward machines for high-level task specification and decomposition in reinforcement learning’. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018, pp. 2107–2116.
- [184] Rodrigo A. Toro Icarte, Toryn Q. Klassen, Richard Valenzano and Sheila A. McIlraith. ‘Reward machines: Exploiting reward function structure in reinforcement learning’. In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 173–208.
- [185] Rodrigo A. Toro Icarte, Ethan Waldie, Toryn Q. Klassen, Rick Valenzano, Margarita Castro and Sheila A. McIlraith. ‘Learning Reward Machines for Partially Observable Reinforcement Learning’. In: *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*. 2019.
- [186] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulao, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun J. Tai, Hannah Tan and Omar G. Younis. *Gymnasium: A standard interface for reinforcement learning environments*. arXiv preprint arXiv:2407.17032. 2024.

- [187] Alessandro Trapasso and Anders Jonsson. ‘Concurrent Multiagent Reinforcement Learning with Reward Machines’. In: *Proceedings of 28th European Conference on Artificial Intelligence (ECAI)*. 2025, pp. 3735–3742.
- [188] John N. Tsitsiklis and Benjamin Van Roy. ‘An analysis of temporal-difference learning with function approximation’. In: *IEEE Transactions on Automatic Control* 42.5 (1997), pp. 674–690.
- [189] Elena Umili, Francesco Argenziano, Aymeric Barbin, Roberto Capobianco et al. ‘Visual reward machines’. In: *Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*. 2023, pp. 255–267.
- [190] Elena Umili, Francesco Argenziano and Roberto Capobianco. *Neural Reward Machines*. arXiv preprint arXiv:2408.08677. 2024.
- [191] Pashootan Vaezipoor, Andrew C. Li, Rodrigo A. Toro Icarte and Sheila A. McIlraith. ‘LTL2Action: Generalizing LTL Instructions for Multi-Task RL’. In: *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 2021, pp. 10497–10508.
- [192] Giovanni Varricchione, Natasha Alechina, Mehdi Dastani, Giuseppe De Giacomo, Brian Logan and Giuseppe Perelli. ‘Pure-Past Action Masking’. In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Technical Track on Safe, Robust and Responsible AI. 2024, pp. 21646–21655.
- [193] Giovanni Varricchione, Natasha Alechina, Mehdi Dastani and Brian Logan. ‘Synthesising Reward Machines for Cooperative Multi-Agent Reinforcement Learning’. In: *Proceedings of the 20th European Conference on Multi-Agent Systems (EUMAS)*. 2023, pp. 328–344.
- [194] Giovanni Varricchione, Natasha Alechina, Mehdi Dastani and Brian Logan. ‘Maximally Permissive Reward Machines’. In: *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI)*. 2024, pp. 1181–1188.
- [195] Giovanni Varricchione, Natasha Alechina, Mehdi Dastani and Brian Logan. ‘Synthesising Reward Machines for Cooperative Multi-Agent Reinforcement Learning’. In: *Journal of Artificial Intelligence Research* 85.38 (2026).
- [196] Giovanni Varricchione, Toryn Q. Klassen, Natasha Alechina, Mehdi Dastani, Brian Logan and Sheila A. McIlraith. ‘Pushdown Reward Machines for Reinforcement Learning’. In: *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 2025, pp. 566–576.
- [197] Akifumi Wachi, Xun Shen and Yanan Sui. ‘A Survey of Constraint Formulations in Safe Reinforcement Learning’. In: *Proceedings of the 33rd International Joint Conference on Artificial Intelligence, (IJCAI)*. Survey Track. 2024, pp. 8262–8271.
- [198] Christopher J. C. H. Watkins. ‘Learning from delayed rewards’. PhD thesis. King’s College, Cambridge (United Kingdom), 1989.

- [199] Christopher J. C. H. Watkins and Peter Dayan. ‘Q-learning’. In: *Machine learning* 8.3 (1992), pp. 279–292.
- [200] Min Wen, Rüdiger Ehlers and Ufuk Topcu. ‘Correct-by-synthesis reinforcement learning with temporal logic constraints’. In: *Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 4983–4990.
- [201] Peter Wieland and Frank Allgöwer. ‘Constructive safety using control barrier functions’. In: *IFAC Proceedings Volumes* 40.12 (2007), pp. 462–467.
- [202] Ronald J. Williams. ‘Simple statistical gradient-following algorithms for connectionist reinforcement learning’. In: *Machine learning* 8.3 (1992), pp. 229–256.
- [203] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu and Bo Wu. ‘Joint Inference of Reward Machines and Policies for Reinforcement Learning’. In: *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*. 2020, pp. 590–598.
- [204] Fangkai Yang, Daoming Lyu, Bo Liu and Steven Gustafson. ‘PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making’. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2018, pp. 4860–4866.
- [205] Qisong Yang, Thiago D. Simão, Simon H. Tindemans and Matthijs T. J. Spaan. ‘Safety-Constrained Reinforcement Learning with a Distributional Safety Critic’. In: *Machine Learning* 112.3 (2023), pp. 859–887.
- [206] Chao Yu, Jiming Liu and Shamim Nemati. *Reinforcement Learning in Healthcare: A Survey*. arXiv preprint arXiv:1908.08796. 2020.
- [207] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals and Peter Battaglia. *Relational Deep Reinforcement Learning*. arXiv preprint arXiv:1806.01830. 2018.
- [208] Alex Zhavoronkov, Yan A. Ivanenkov, Alex Aliper, Mark S. Veselov, Vladimir A. Aladinskiy, Anastasiya V. Aladinskaya, Victor A. Terentiev, Daniil A. Polykovskiy, Maksim D. Kuznetsov, Arip Asadulaev, Yury Volkov, Artem Zholus, Rim R. Shayakhmetov, Alexander Zhebrak, Lidiya I. Minaeva, Bogdan A. Zagribelnyy, Lennart H. Lee, Richard Soll, David Madge, Li Xing, Tao Guo and Alán Aspuru-Guzik. ‘Deep learning enables rapid identification of potent DDR1 kinase inhibitors’. In: *Nature biotechnology* 37.9 (2019), pp. 1038–1040.
- [209] Xuejing Zheng and Chao Yu. *Multi-Agent Reinforcement Learning with a Hierarchy of Reward Machines*. arXiv preprint arXiv:2403.07005. 2024.

- [210] Zhenpeng Zhou, Xiaocheng Li and Richard N. Zare. ‘Optimizing chemical reactions with deep reinforcement learning’. In: *ACS Central Science* 3.12 (2017), pp. 1337–1344.
- [211] Chenyang Zhu, Jinyu Zhu, Wen Si, Xueyuan Wang and Fang Wang. ‘Multi-agent reinforcement learning with synchronized and decomposed reward automaton synthesized from reactive temporal logic’. In: *Knowledge-Based Systems* 306 (2024), 112703 (1–16).
- [212] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu and Moshe Y. Vardi. ‘Symbolic LTLf Synthesis’. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. 2017, pp. 1362–1369.
- [213] Lenore Zuck. ‘Past temporal logic’. PhD thesis. The Weizmann Institute of Science, Rehovot (Israel), 1986.



Samenvatting

Reinforcement learning (RL) is een machine learning-paradigma waarbij KI-agenten worden getraind om een taak uit te voeren via trial-and-error, om zo opgebouwde beloningssignalen te maximaliseren. Er worden doorgaans twee aannames gedaan binnen de reinforcement learning: (i) dat zowel wij als de agent geen a-priori kennis hebben van de omgeving waarin de agent zal handelen en welke beloningssignalen deze zal waarnemen, en (ii) dat de omgeving waarin de agent handelt *markoviaans* is, wat betekent dat de huidige toestand en de actie van de agent voldoende zijn om de kansverdeling voor de volgende toestand te bepalen. Hoewel deze twee aannames reinforcement learning tot een breed inzetbaar raamwerk voor het trainen van agenten maken, is het moeilijk om agenten te trainen zich aan veiligheidseisen te houden, aangezien reinforcement learning geen garanties biedt dat de agent dergelijke beperkingen nooit zal schenden. Bovendien gaan beide aannames in veel gevallen niet op. Een ontwerper kan a-priori kennis hebben van de taak en de definities van de omgeving, die dus gebruikt zouden kunnen worden om de lerende agent te informeren. De markoviaanse aanname geldt mogelijk niet als de taak bijvoorbeeld vereist dat de agent in de loop van de tijd specifieke reeksen acties uitvoert.

In dit proefschrift gebruiken we technieken en ideeën uit declaratieve KI om nieuwe, efficiëntere en veiligere methoden voor reinforcement learning te definiëren wanneer de twee eerder besproken aannames niet opgaan. Declaratieve KI omvat alle methoden en technieken waarbij KI-agenten a-priori worden gedefinieerd met behulp van bijvoorbeeld logica, planalgoritmes of regels, om te bepalen hoe ze zullen handelen. In tegenstelling tot reinforcement learning zijn methoden van declaratieve KI meer geschikt wanneer ontwerpers de omgevingen kennen waarin de agent zal proberen zijn taak te volbrengen. Verder kan men door het gebruik van technieken zoals logica en planalgoritmes eenvoudig regels definiëren die de agent kunnen helpen bij het volbrengen van taken die complex, niet-markoviaans gedrag vereisen. Ten slotte is het, doordat methoden uit de declaratieve KI hulpmiddelen en ideeën uit de formele wiskunde gebruiken, eenvoudiger om methoden te ontwikkelen die garanderen dat de agent zich aan veiligheidseisen houdt.

We presenteren eerst nieuwe methoden die agenten helpen om niet-markoviaanse taken op een efficiëntere manier te leren en om beter te presteren. Hiervoor gebruiken we beloningsmachines (*reward machines*), een formalisme dat recent in de literatuur over reinforcement learning is voorgesteld, en dat kan worden gebruikt om agenten te trainen in het volbrengen van niet-markoviaanse taken. We laten eerst zien hoe beloningsmachines automatisch kunnen worden gebouwd door gebruik te maken van planalgoritmes. Verder zullen de verkregen beloningsmachines “*maximaal permissief*” zijn, wat inhoudt dat de agent alle mogelijke manieren leert om de taak te volbrengen. Vervolgens laten we zien hoe het door gebruik van alternerende-tijd temporele logica (*Alternating-time Temporal Logic*, ATL) mogelijk is om beloningsmachines te verkrijgen die gebruikt kunnen worden om coalities van KI-agenten te trainen om een gezamenlijke taak te volbrengen. We specificeren de taak van de coalitie met een ATL-formule en verkrijgen door middel van modelverificatie een beloningsmachines die gebruikt kan worden om de coalitie te trainen in het volbrengen van de taak. Ten slotte breiden we beloningsmachines uit met een stapel, waardoor we *stapelbeloningsmachines* (*pushdown reward machines*) verkrijgen. Aangezien stapelbeloningsmachines gebaseerd zijn op stapels, kunnen ze worden gebruikt om agenten te trainen in het volbrengen van complexere niet-markoviaanse taken in vergelijking met standaard beloningsmachines.

Vervolgens bieden we een nieuwe methode om agenten te trainen en de naleving van veiligheidseisen te garanderen, zowel tijdens als na de training. Om dit te doen, maken we gebruik van lineaire-tijd temporele logic met uitsluitend verleden-operatoren (*Pure-Past Linear-time Temporal Logic*, PPLTL). In onze methode wordt elke actie a die de agent uit kan voeren geassocieerd met een PPLTL-formule φ_a . We staan de agent vervolgens toe om op een gegeven moment een actie a uit te voeren, enkel indien de bijbehorende PPLTL-formule φ_a op dat moment waar is. Aangezien het voldoen aan de formule φ_a impliceert dat de bijbehorende actie a op dat moment veilig is, zal de agent nooit een actie uitvoeren die een veiligheidseis schendt. De methode garandeert dus dat de agent nooit een veiligheidseis zal schenden.

We concluderen het proefschrift met ideeën voor nieuwe onderzoeksrichtingen op basis van de resultaten en methoden die we presenteren. Dit omvat het overwegen van praktische toepassingen in de echte wereld, complexere omgevingen waarin de agent niet alle relevante informatie omtrent de toestand van de omgeving kan waarnemen, en omgevingen waarin de agent meerdere doelen tegelijkertijd moet behalen.



SIKS Dissertation Series

-
- 2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UVA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UVA), Context & Semantics in News & Web Search

- 21 Alejandro Moreno Céleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
- 22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
- 23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
- 24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
- 25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
- 26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
- 27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
- 28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
- 29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
- 30 Ruud Mattheij (UvT), The Eyes Have It
- 31 Mohammad Khelghati (UT), Deep web content monitoring
- 32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
- 33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
- 34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
- 35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
- 36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
- 37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
- 38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
- 39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
- 40 Christian Detweiler (TUD), Accounting for Values in Design
- 41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
- 42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
- 43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
- 44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
- 45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
- 46 Jorge Gallego Perez (UT), Robots to Make you Happy

-
- 47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
 - 48 Tanja Buttler (TUD), Collecting Lessons Learned
 - 49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
 - 50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains
-
- 2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
 - 02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
 - 03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
 - 04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
 - 05 Mahdieh Shadi (UVA), Collaboration Behavior
 - 06 Damir Vandić (EUR), Intelligent Information Systems for Web Product Search
 - 07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
 - 08 Rob Konijn (VU) , Detecting Interesting Differences: Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
 - 09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
 - 10 Robby van Delden (UT), (Steering) Interactive Play Behavior
 - 11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
 - 12 Sander Leemans (TUE), Robust Process Mining with Guarantees
 - 13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
 - 14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
 - 15 Peter Berck (RUN), Memory-Based Text Correction
 - 16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
 - 17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
 - 18 Ridho Reinanda (UVA), Entity Associations for Search
 - 19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
 - 20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
 - 21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
 - 22 Sara Magliacane (VU), Logics for causal inference under uncertainty
 - 23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
 - 24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning

- 25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
 - 26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
 - 27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
 - 28 John Klein (VU), Architecture Practices for Complex Contexts
 - 29 Adel Alhuraibi (UvT), From IT-Business Strategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
 - 30 Wilma Latuny (UvT), The Power of Facial Expressions
 - 31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
 - 32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
 - 33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
 - 34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
 - 35 Martine de Vos (VU), Interpreting natural science spreadsheets
 - 36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
 - 37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
 - 38 Alex Kayal (TUD), Normative Social Applications
 - 39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
 - 40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
 - 41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
 - 42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
 - 43 Maaïke de Boer (RUN), Semantic Mapping in Video Retrieval
 - 44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
 - 45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
 - 46 Jan Schneider (OU), Sensor-based Learning Support
 - 47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
 - 48 Angel Suarez (OU), Collaborative inquiry-based learning
-
- 2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
 - 02 Felix Mannhardt (TUE), Multi-perspective Process Mining
 - 03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction

- 04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
 - 05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
 - 06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
 - 07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
 - 08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
 - 09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
 - 10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
 - 11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
 - 12 Xixi Lu (TUE), Using behavioral context in process mining
 - 13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
 - 14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
 - 15 Naser Davarzani (UM), Biomarker discovery in heart failure
 - 16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
 - 17 Jianpeng Zhang (TUE), On Graph Sample Clustering
 - 18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
 - 19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
 - 20 Manxia Liu (RUN), Time and Bayesian Networks
 - 21 Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
 - 22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
 - 23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
 - 24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
 - 25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
 - 26 Roelof Anne Jelle de Vries (UT), Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
 - 27 Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
 - 28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
 - 29 Yu Gu (UVT), Emotion Recognition from Mandarin Speech
 - 30 Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
-
- 2019 01 Rob van Eijk (UL), Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
- 02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty

- 03 Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
- 04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
- 05 Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
- 06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
- 07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
- 08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
- 09 Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
- 10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
- 11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
- 12 Jacqueline Heinerman (VU), Better Together
- 13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
- 14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
- 15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
- 16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
- 17 Ali Hurriyetoglu (RUN), Extracting actionable information from microtexts
- 18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
- 19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
- 20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
- 21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
- 22 Martin van den Berg (VU), Improving IT Decisions with Enterprise Architecture
- 23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
- 24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
- 25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
- 26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
- 27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
- 28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
- 29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances

-
- 30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
 - 31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
 - 32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
 - 33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
 - 34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
 - 35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
 - 36 Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
 - 37 Jian Fang (TUD), Database Acceleration on FPGAs
 - 38 Akos Kadar (OUN), Learning visually grounded and multilingual representations
-
- 2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
 - 02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
 - 03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
 - 04 Maarten van Gompel (RUN), Context as Linguistic Bridges
 - 05 Yulong Pei (TUE), On local and global structure mining
 - 06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
 - 07 Wim van der Vegt (OUN), Towards a software architecture for reusable game components
 - 08 Ali Mirsoleimani (UL), Structured Parallel Programming for Monte Carlo Tree Search
 - 09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
 - 10 Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
 - 11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data Augmentation- Methods for Long-Tail Entity Recognition Models
 - 12 Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
 - 13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
 - 14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
 - 15 Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
 - 16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
 - 17 Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences

- 18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
 - 19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
 - 20 Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
 - 21 Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be
 - 22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
 - 23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
 - 24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
 - 25 Xin Du (TUE), The Uncertainty in Exceptional Model Mining
 - 26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
 - 27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
 - 28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
 - 29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
 - 30 Bob Zadok Blok (UL), Creatief, Creatieve, Creatiefst
 - 31 Gongjin Lan (VU), Learning better – From Baby to Better
 - 32 Jason Rhuggenaath (TUE), Revenue management in online markets: pricing and online advertising
 - 33 Rick Gilsing (TUE), Supporting service-dominant business model evaluation in the context of business model innovation
 - 34 Anna Bon (MU), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
 - 35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
-
- 2021 01 Francisco Xavier Dos Santos Fonseca (TUD), Location-based Games for Social Interaction in Public Space
 - 02 Rijk Mercur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
 - 03 Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with Smart Devices
 - 04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
 - 05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
 - 06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
 - 07 Armel Lefebvre (UU), Research data management for open science
 - 08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking

-
- 09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
 - 10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
 - 11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
 - 12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
 - 13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
 - 14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
 - 15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
 - 16 Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
 - 17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
 - 18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
 - 19 Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
 - 20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
 - 21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
 - 22 Sihang Qiu (TUD), Conversational Crowdsourcing
 - 23 Hugo Manuel Proença (LIACS), Robust rules for prediction and description
 - 24 Kaijie Zhu (TUE), On Efficient Temporal Subgraph Query Processing
 - 25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
 - 26 Benno Kruit (CWI & VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
 - 27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
 - 28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs
-
- 2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
 - 02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
 - 03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
 - 04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
 - 05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
 - 06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding

- 07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
- 08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
- 09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
- 10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
- 11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
- 12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
- 13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
- 14 Michiel Overeem (UU), Evolution of Low-Code Platforms
- 15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
- 16 Pieter Gijbbers (TU/e), Systems for AutoML Research
- 17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
- 18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
- 19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
- 20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
- 21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
- 22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
- 23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
- 24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
- 25 Anna L.D. Latour (LU), Optimal decision-making under constraints and uncertainty
- 26 Anne Dirkson (LU), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
- 27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
- 28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
- 29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
- 30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays

-
- 31 Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
 - 32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
 - 33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
 - 34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
 - 35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction
-
- 2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
 - 02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
 - 03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
 - 04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
 - 05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
 - 06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
 - 07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
 - 08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
 - 09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
 - 10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
 - 11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
 - 12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
 - 13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
 - 14 Selma Čaušević (TUD), Energy resilience through self-organization
 - 15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
 - 16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
 - 17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
 - 18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation

- 19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
 - 20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
 - 21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
 - 22 Alireza Shojaifar (UU), Volitional Cybersecurity
 - 23 Theo Theunissen (UU), Documentation in Continuous Software Development
 - 24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
 - 25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
 - 26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
 - 27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions
 - 28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
 - 29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results
-
- 2024
- 01 Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education
 - 02 Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems
 - 03 Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis
 - 04 Mike Huisman (UL), Understanding Deep Meta-Learning
 - 05 Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair
 - 06 Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence
 - 07 Parisa Shayan (TiU), Modeling User Behavior in Learning Management Systems
 - 08 Xin Zhou (UvA), From Empowering to Motivating: Enhancing Policy Enforcement through Process Design and Incentive Implementation
 - 09 Giso Dal (UT), Probabilistic Inference Using Partitioned Bayesian Networks
 - 10 Cristina-Iulia Bucur (VUA), Linkflows: Towards Genuine Semantic Publishing in Science
 - 11 withdrawn
 - 12 Peide Zhu (TUD), Towards Robust Automatic Question Generation For Learning
 - 13 Enrico Liscio (TUD), Context-Specific Value Inference via Hybrid Intelligence
 - 14 Larissa Capobianco Shimomura (TU/e), On Graph Generating Dependencies and their Applications in Data Profiling

- 15 Ting Liu (VUA), A Gut Feeling: Biomedical Knowledge Graphs for Interrelating the Gut Microbiome and Mental Health
- 16 Arthur Barbosa C mara (TUD), Designing Search-as-Learning Systems
- 17 Razieh Alidoosti (VUA), Ethics-aware Software Architecture Design
- 18 Laurens Stoop (UU), Data Driven Understanding of Energy-Meteorological Variability and its Impact on Energy System Operations
- 19 Azadeh Mozafari Mehr (TU/e), Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior
- 20 Ritsart Anne Plantenga (UL), Omgang met Regels
- 21 Federica Vinella (UU), Crowdsourcing User-Centered Teams
- 22 Zeynep Ozturk Yurt (TU/e), Beyond Routine: Extending BPM for Knowledge-Intensive Processes with Controllable Dynamic Contexts
- 23 Jie Luo (VUA), Lamarck's Revenge: Inheritance of Learned Traits Improves Robot Evolution
- 24 Nirmal Roy (TUD), Exploring the effects of interactive interfaces on user search behaviour
- 25 Alisa Rieger (TUD), Striving for Responsible Opinion Formation in Web Search on Debated Topics
- 26 Tim Gubner (CWI), Adaptively Generating Heterogeneous Execution Strategies using the VOILA Framework
- 27 Lincen Yang (UL), Information-theoretic Partition-based Models for Interpretable Machine Learning
- 28 Leon Helwerda (UL), Grip on Software: Understanding development progress of Scrum sprints and backlogs
- 29 David Wilson Romero Guzman (VUA), The Good, the Efficient and the Inductive Biases: Exploring Efficiency in Deep Learning Through the Use of Inductive Biases
- 30 Vijanti Ramautar (UU), Model-Driven Sustainability Accounting
- 31 Ziyu Li (TUD), On the Utility of Metadata to Optimize Machine Learning Workflows
- 32 Vinicius Stein Dani (UU), The Alpha and Omega of Process Mining
- 33 Siddharth Mehrotra (TUD), Designing for Appropriate Trust in Human-AI interaction
- 34 Robert Deckers (VUA), From Smallest Software Particle to System Specification - MuDForM: Multi-Domain Formalization
- 35 Sicui Zhang (TU/e), Methods of Detecting Clinical Deviations with Process Mining: a Fuzzy Set Approach
- 36 Thomas Mulder (TU/e), Optimization of Recursive Queries on Graphs
- 37 James Graham Nevin (UvA), The Ramifications of Data Handling for Computational Models
- 38 Christos Koutras (TUD), Tabular Schema Matching for Modern Settings
- 39 Paola Lara Machado (TU/e), The Nexus between Business Models and Operating Models: From Conceptual Understanding to Actionable Guidance
- 40 Montijn van de Ven (TU/e), Guiding the Definition of Key Performance Indicators for Business Models
- 41 Georgios Siachamis (TUD), Adaptivity for Streaming Dataflow Engines

- 42 Emmeke Veltmeijer (VUA), Small Groups, Big Insights: Understanding the Crowd through Expressive Subgroup Analysis
 - 43 Cedric Waterschoot (KNAW Meertens Instituut), The Constructive Conundrum: Computational Approaches to Facilitate Constructive Commenting on Online News Platforms
 - 44 Marcel Schmitz (OU), Towards learning analytics-supported learning design
 - 45 Sara Salimzadeh (TUD), Living in the Age of AI: Understanding Contextual Factors that Shape Human-AI Decision-Making
 - 46 Georgios Stathis (Leiden University), Preventing Disputes: Preventive Logic, Law & Technology
 - 47 Daniel Daza (VUA), Exploiting Subgraphs and Attributes for Representation Learning on Knowledge Graphs
 - 48 Ioannis Petros Samiotis (TUD), Crowd-Assisted Annotation of Classical Music Compositions
-
- 2025 01 Max van Haastrecht (UL), Transdisciplinary Perspectives on Validity: Bridging the Gap Between Design and Implementation for Technology-Enhanced Learning Systems
 - 02 Jurgen van den Hoogen (JADS), Time Series Analysis Using Convolutional Neural Networks
 - 03 Andra-Denis Ionescu (TUD), Feature Discovery for Data-Centric AI
 - 04 Rianne Schouten (TU/e), Exceptional Model Mining for Hierarchical Data
 - 05 Nele Albers (TUD), Psychology-Informed Reinforcement Learning for Situated Virtual Coaching in Smoking Cessation
 - 06 Daniël Vos (TUD), Decision Tree Learning: Algorithms for Robust Prediction and Policy Optimization
 - 07 Ricky Maulana Fajri (TU/e), Towards Safer Active Learning: Dealing with Unwanted Biases, Graph-Structured Data, Adversary, and Data Imbalance
 - 08 Stefan Bloemheuvel (TiU), Spatio-Temporal Analysis Through Graphs: Predictive Modeling and Graph Construction
 - 09 Fadime Kaya (VUA), Decentralized Governance Design - A Model-Based Approach
 - 10 Zhao Yang (UL), Enhancing Autonomy and Efficiency in Goal-Conditioned Reinforcement Learning
 - 11 Shahin Sharifi Noorian (TUD), From Recognition to Understanding: Enriching Visual Models Through Multi-Modal Semantic Integration
 - 12 Lijun Lyu (TUD), Interpretability in Neural Information Retrieval
 - 13 Fuda van Diggelen (VUA), Robots Need Some Education: on the complexity of learning in evolutionary robotics
 - 14 Gennaro Gala (TU/e), Probabilistic Generative Modeling with Latent Variable Hierarchies
 - 15 Michiel van der Meer (UL), Opinion Diversity through Hybrid Intelligence
 - 16 Monika Grewal (TU Delft), Deep Learning for Landmark Detection, Segmentation, and Multi-Objective Deformable Registration in Medical Imaging

- 17 Matteo De Carlo (VUA), Real Robot Reproduction: Towards Evolving Robotic Ecosystems
- 18 Anouk Neerinx (UU), Robots That Care: How Social Robots Can Boost Children's Mental Wellbeing
- 19 Fang Hou (UU), Trust in Software Ecosystems
- 20 Alexander Melchior (UU), Modelling for Policy is More Than Policy Modelling (The Useful Application of Agent-Based Modelling in Complex Policy Processes)
- 21 Mandani Ntekouli (UM), Bridging Individual and Group Perspectives in Psychopathology: Computational Modeling Approaches using Ecological Momentary Assessment Data
- 22 Hilde Weerts (TU/e), Decoding Algorithmic Fairness: Towards Interdisciplinary Understanding of Fairness and Discrimination in Algorithmic Decision-Making
- 23 Roderick van der Weerd (VUA), IoT Measurement Knowledge Graphs: Constructing, Working and Learning with IoT Measurement Data as a Knowledge Graph
- 24 Zhong Li (UL), Trustworthy Anomaly Detection for Smart Manufacturing
- 25 Kyana van Eijndhoven (TiU), A Breakdown of Breakdowns: Multi-Level Team Coordination Dynamics under Stressful Conditions
- 26 Tom Pepels (UM), Monte-Carlo Tree Search is Work in Progress
- 27 Danil Provodin (JADS, TU/e), Sequential Decision Making Under Complex Feedback
- 28 Jinke He (TUD), Exploring Learned Abstract Models for Efficient Planning and Learning
- 29 Erik van Haeringen (VUA), Mixed Feelings: Simulating Emotion Contagion in Groups
- 30 Myrthe Reuver (VUA), A Puzzle of Perspectives: Interdisciplinary Language Technology for Responsible News Recommendation
- 31 Gebrekirstos Gebreselassie Gebremeskel (RUN), Spotlight on Recommender Systems: Contributions to Selected Components in the Recommendation Pipeline
- 32 Ryan Brate (UU), Words Matter: A Computational Toolkit for Charged Terms
- 33 Merle Reimann (VUA), Speaking the Same Language: Spoken Capability Communication in Human-Agent and Human-Robot Interaction
- 34 Eduard C. Groen (UU), Crowd-Based Requirements Engineering
- 35 Urja Khurana (VUA), From Concept To Impact: Toward More Robust Language Model Deployment
- 36 Anna Maria Wegmann (UU), Say the Same but Differently: Computational Approaches to Stylistic Variation and Paraphrasing
- 37 Chris Kamphuis (RUN), Exploring Relations and Graphs for Information Retrieval
- 38 Valentina Maccatrozzo (VUA), Break the Bubble: Semantic Patterns for Serendipity

- 39 Dimitrios Alivanistos (VUA), Knowledge Graphs & Transformers for Hypothesis Generation: Accelerating Scientific Discovery in the Era of Artificial Intelligence
- 40 Stefan Grafberger (UvA), Declarative Machine Learning Pipeline Management via Logical Query Plans
- 41 Mozghan Vazifehdoostirani (TU/e), Leveraging Process Flexibility to Improve Process Outcome - From Descriptive Analytics to Actionable Insights
- 42 Margherita Martorana (VUA), Semantic Interpretation of Dataless Tables: a metadata-driven approach for findable, accessible, interoperable and reusable restricted access data
- 43 Krist Shingjergji (OU), Sense the Classroom - Using AI to Detect and Respond to Learning-Centered Affective States in Online Education
- 44 Robbert Reijnen (TU/e), Dynamic Algorithm Configuration for Machine Scheduling Using Deep Reinforcement Learning
- 45 Anjana Mohandas Sheeladevi (VUA), Occupant-Centric Energy Management: Balancing Privacy, Well-being and Sustainability in Smart Buildings
- 46 Ya Song (TU/e), Graph Neural Networks for Modeling Temporal and Spatial Dimensions in Industrial Decision-making
- 47 Tom Kouwenhoven (UL), Collaborative Meaning-Making. The Emergence of Novel Languages in Humans, Machines, and Human-Machine Interactions
- 48 Evy van Weelden (TiU), Integrating Virtual Reality and Neurophysiology in Flight Training
- 49 Selene Báez Santamaría (VUA), Knowledge-centered conversational agents with a drive to learn
- 50 Lea Krause (VUA), Contextualising Conversational AI
- 51 Jiaxu Zhao (TU/e), Understanding and Mitigating Unwanted Biases in Generative Language Models
- 52 Qiao Xiao (TU/e), Model, Data and Communication Sparsity for Efficient Training of Neural Networks
- 53 Gaole He (TUD), Towards Effective Human-AI Collaboration: Promoting Appropriate Reliance on AI Systems
- 54 Go Sugimoto (VUA), MISSING LINKS Investigating the Quality of Linked Data and its Tools in Cultural Heritage and Digital Humanities
- 55 Sietze Kai Kuilman (TUD), AI that Glitters is Not Gold: Requirements for Meaningful Control of AI Systems
- 56 Wijnand van Woerkom (UU), A Fortiori Case-Based Reasoning: Formal Studies with Applications in Artificial Intelligence and Law
- 57 Syeda Amna Sohail (UT), Privacy-Utility Trade-Off in Healthcare Metadata Sharing and Beyond: A Normative and Empirical Evaluation at Inter and Intra Organizational Levels
- 58 Junhan Wen (TUD), "From iMage to Market": Machine-Learning-Empowered Fruit Supply
- 59 Mohsen Abbaspour Onari (TU/e), From Explanation to Trust: Modeling and Measuring Trust in Explainable Decision Support

-
- 60 Marcel Jurriaan Robeer (UU), Beyond Trust: A Causal Approach to Explainable AI in Law Enforcement
 - 61 Shuai Wang (VUA), Links in Large Integrated Knowledge Graphs: Analysis, Refinement, and Domain Applications
 - 62 Khaleel Asyraaf Mat Sanusi (OU), Augmenting a learning model within immersive learning environments for psychomotor skills
 - 63 Rashid Zaman (TU/e), Online Conformance Checking on Degraded Data
 - 64 Jens d'Hondt (TU/e), Effective and Efficient Multivariate Similarity Search
 - 65 Aswin Balasubramaniam (UT), Disentangling Runner Drone Interaction Potentialities
-
- 2026 01 Pei-Yu Chen (TUD), Human-Agent Alignment Dialogues: Eliciting User Information at Runtime for Personalized Behavior Support
 - 02 Hezha Hassan Mohammedkhan (TiU), Estimating Body Measurements of Children from 2D Images: Towards the Automatic Detection of Malnutrition
 - 03 Kyriakos Psarakis (TUD), Democratizing Scalable Cloud Applications: Transactional Stateful Functions on Streaming Dataflows
 - 04 Boyu Xu (UU), Exploring Indirect Relations between Topics in Neuroscience Literature Using Augmented Reality to Inform Experimental Design
 - 05 Koen Minartz (TU/e), Stochastic Simulation with Geometric Deep Generative Models
 - 06 Azim Afroozeh (CWI, VUA), FastLanes: a Next-Gen File Format
 - 07 Inès Blin (VUA), Narrative Understanding with Knowledge Graphs
 - 08 Paul van Vulpen (UU), Debating Digital Dominance: Decentralized Technology Governance for Strategic Autonomy
 - 09 Afrizal Doewes (TU/e), Rethinking Automated Essay Scoring: Agreement, Fairness, and Feedback
 - 10 Nikolaos Delapaschos Kondylidis (VUA), Establishing Task-Oriented Understanding between Agents
 - 11 Işıl Baysal Erez (UT), Handling Missing Data with Meta-Learning and Large Language Models
 - 12 Xue Li (UvA), From Fine-tuning to Prompting: a Paradigm Shift in Knowledge Graph Construction
 - 13 Isaac da Silva Torres (VUA), Guidelines to Flux Between Conceptual Models: Understanding Complex Digital Business Ecosystems
 - 14 Philip Lippmann (TUD), Synthetic Data for Robust Language Modelling
 - 15 Rashmi Khazanchi (OU), Artificial Intelligence in Education: Impact of AI-Based Systems on Mathematics Achievement
 - 16 Carolina Ferreira Gomes Centeio Jorge (TUD), Modelling Artificial Trust for Effective Human-AI Teamwork
 - 17 Maria Tsfasman (TUD), Towards Predicting Memory in Multimodal Group Interactions
 - 18 Riccardo Lo Bianco (TU/e), Deep Reinforcement Learning for Automated Decision-Making in Process Management Systems
 - 19 Israel Campero Jurado (TU/e), Innovations in Optimization and Applications in Healthcare

- 20 Iftitahu Ni'mah (TU/e), Contrastive Learning and Evaluation in Low Resource Scenario of Natural Language Processing
 - 21 Francisco N.F.Q. Simoes (UU), Causality, Information, and Decision-Making
 - 22 Ruben Verhagen (TUD), Transparent and Explainable Agents for Human-Agent Teaming
 - 23 Eduardo Calò (UU), Automatically Expressing the Meaning of Logical Formulae in Natural Language
 - 24 Shuai Han (UU), Improving Sample Efficiency of Reinforcement Learning: Exploiting Structural Knowledge for Decision Making
 - 25 Francis Saa-Dittoh (VUA), From Radio to AI: African Community-Driven Development of Sustainable Information Systems
 - 26 Mohammed Al Owayyed (TUD), Interactive Simulation-Based Learning Tools for Training Children's Helpline Counsellors
 - 27 Bram Grooten (TU/e), Adaptive Reinforcement Learning: Lean and Dynamic Agents for Robust Generalization
 - 28 Anouck Braggaar (TiU), Does This Answer Your question? Evaluating, Replicating, and Improving Task-Oriented Dialogue Systems
 - 29 Afsana Khan (UM), Unlocking the Value of Data with Vertical Federated Learning
 - 30 Yucheng Yang (TU/e), Generalization in Reinforcement Learning: Theories and Algorithms for Downstream Task and Multi-objective Trade-off Adaptation
 - 31 Luis Pedro Silvestrin (VUA), Efficient Machine Learning for Time-Varying Data: Contributions to Time-Series Machine Learning and Transfer Learning
-



Curriculum Vitae

Research Experience

Universiteit Utrecht

Mar 2021 - Feb 2026

PhD in Efficient and Safe Reinforcement Learning

- Supervised by Prof. Dr. Mehdi Dastani and Prof. Dr. Brian Logan
- Various teaching duties: several supervisions of bachelor's and master's students, supervision of tutorials and project groups, co-lecturing

Education

Universiteit van Amsterdam

Sep 2019 - May 2021

MSc in Logic (cum laude)

- Thesis: Complexity of Locally Fair Allocations on Graphs
- GPA: 8.98/10, thesis grade: 9/10
- Focus on computability, reasoning, computational social choice and game theory

Università degli Studi di Roma "La Sapienza"

Sep 2018 - Aug 2019

MSc in Computer Science (transferred)

- Classes completed: Models of Computation, Machine Learning, Mathematical Logic for Computer Science, Biometric Systems, Natural Language Processing

Università degli Studi di Roma "La Sapienza"

Sep 2015 - Oct 2018

BSc in Computer Science (cum laude)

- Thesis: Automatic Versions of Combinatorics Theorems
- Final grade: 110/110



Acknowledgements

My first thanks go to my supervisors, Brian, Mehdi, and Natasha. Natasha, I still remember when I contacted you asking for more information about this PhD position. You were enthusiastic to talk about it with me, and you helped me through the process of writing a project proposal. Although you have not been an official supervisor of mine since the latter half of my first year, you have not stopped supporting and helping me grow in these years. Because of this, I will always be grateful. Plus, you took several pictures of me giving talks, which I always found extremely cute. Mehdi, you have taught me how to carefully present my ideas, and were a great “opponent” during our meetings. You had me look at my approaches and problems from different perspectives, which I think is always good practice in science. You would also not shy away from giving compliments, and checking in when I was not feeling the best. Thanks also for involving me in the DECIDE project and, most of all, for being a great mentor. Finally, thanks Brian for your thorough supervision. Throughout our meetings you have always carefully checked that I was remaining focused on my PhD research, while still giving me the freedom of choosing which directions to pursue. Most of all, you were patient enough to help me understand what my research was really about, especially when I was very lost and was not even sure whether I should have done a PhD in reinforcement learning at all. I suppose that took quite some patience, and of that I am deeply thankful.

My next thanks have to go to Nima. While technically we met during the MoL (you’ll probably confidently say that you remember me since the first step I took in the ILLC), it was not until Dragan “matched us” that we got to know each other. In Rennes, we then decided that we would share an office, because there were potential research directions that we could pursue together. I think that the lack of published work shows how much I have enjoyed your presence in the office. I will always cherish the days spent looking through websites of questionable people, discovering and re-appreciating memes, listening to music, and discussing whatever we wanted to discuss, especially my Italian (or should I say Persian?) roots and heritage. Thanks for being one of my paranympths, and for being such a great office mate and friend.

My impression is that one does not go through a PhD without fellow PhD students who go through similar doubts, issues, and setbacks, and with whom entire afternoons can be spent talking over snacks and a cup of coffee or tea. Thanks Annet for being so wise, supportive, and easy to ask for advice. Thanks also for being such an avid board game player, and for being so open to spontaneous outings, a quality that I have learned to greatly appreciate in the Netherlands. Jeremy, thanks for supplying me with spicy American sauce for the last two years, and for loving coffee and burgers so much (speaking of, we have not had a burger in a while, and we should fix that as soon as possible). Most of all, thanks for always being there to listen, for being so empathetic, and for being the most wholesome person I know. Tamara, thanks for being such a merry (yes, I am referring to *that* picture) and passionate presence in the office. Thanks, also to Marinka, for helping moving me and Lise to our apartment, for fixing our bed in the blink of an eye, and for gifting us a gorgeous temporary toilet lamp. Sylvia, you were the last person to join the office, but in the years that we have shared it I have grown fond of your singing, dancing, and interest in pretty much everything: never lose this, please! Thanks Anastasia for hosting me in your office and thanks also for your wonderful weirdness. Likewise, thanks Duygu for welcoming me in your office, but also for providing Turkish delights, and for being so sweet and caring. Maksim, you brought Russian stoicism to the office and entertained us with questions about your DNA, thanks for this and for all the drinks we got while travelling. Ira, thanks for indulging me and Nima in and out of the office, and for being a fellow gamerTM. Roos, thanks for your delicious baking treats and never-ending supply of board games. To all of the above, and to Changxi, Claudia, Debargha, Francisco, Enas, Heleen, Hugh Mee, Jan, Jelle, Jurian, Katerina, Kristina, Manzoor, Michele, Panos, Shuai, Thomas, Vahid, and Wijnand: thank you so much for being part of the last five, wonderful, years. Finally, to the new “generation” of PhD students: I hope you will have at least as much fun and learn as much as I have during my PhD. Good luck!

Half of the papers in this thesis were the result of work with external researchers, who have been great collaborators and teachers. Giuseppe (De Giacomo), thanks for hosting me in your group, and for your brilliant insights while working on my second paper. Giuseppe (Perelli), thanks for keeping a spot in your office in Rome always available, for your intuitions in sustainable AI, and for always being so interested and open to talk about research. Sheila, thanks for the help in preparing the last talk of my PhD, and most of all for the (research) wisdom you have imparted on me in the year and a half we have worked together. Finally, thanks Torny for being such a formidable sparring partner and researcher: before our meetings I was always “scared” of you having found some error or imprecision, but I would say that these are the kind of good fears one can have when doing research.

I also had the luck to be surrounded by great colleagues at UU. Dragan, thanks for matching me and Nima, for all the chats in front of your office, and for inviting me to give a talk as an independent researcher. João, I have met you in the last months of my PhD, but you have tried really hard to help me in my job search, and of that I am very grateful. I hope that in the next two years I can repay you for that, at least with a few (as promised) beers. Roxana, though you have joined the department in the latter

half of my PhD, you were quickly part of our daily gatherings and shenanigans. Be it to talk about research, or to grab something after work, it is always great and fun to have you around. As with Mehdi, thanks also for involving me in the DECIDE project: I eagerly look forward to the two years we have to work together. Shihan, first of all thanks for joining the assessment committee and for providing great feedback. Whenever we bumped into each other in the corridor, you always made sure to check how I was doing; thanks for being so caring. Thanks for your tickets to the water park in Tenerife, and finally for always bringing back delicious treats from China. Wouter, thanks for involving me so much in the logic course, and for always being available to answer any question I had about the material or grading. Thanks for helping me with the development document to obtain the BKO; unfortunately, that did not work out in the end, but I am grateful for the time you spent on it. Thanks to Davide, Dirk, Heysem, Janneke, Marco, Pinar, Rumana, and Silja also for being part of such group of colleagues. Finally, thanks to the admin people at the ICT department; particularly to Annette for her help with the Social Events Committee and for helping me get my backpack when it was locked in an office after 5PM, and to Iris for helping organise the S4S Café.

Cecilia, Claudio, and Luca, you made me feel more at home here in Utrecht, and made me realise that I always need a cohort of people from Rome nearby. Luca, thanks for your visits in Utrecht, for having me in your office in Rome, and for all the efforts you put in our work. Cecilia, thanks for the coffees and all the chats we had in the last months. You are one of the most hard-working and joyful people I have ever met, and I look forward to see what the future holds for you. Claudio, thanks for having such a contagious enthusiasm, for trying to involve me in so many activities and projects, and for saving me from looking for a last quorum member. More importantly (/s, but not so much), thanks for being a fellow romanista at UU.

I have had the pleasure of meeting lovely people from academia, so here is a thought for them. Thanks to the members of the assessment committee for deciding to take the time to read this thesis and evaluate it. Thanks to those from the ILLC and from *Bahareh's gang*: Daniël, Johannes, Marianna, Mina, Lide, Lydia, Raheleh, Robin, Rodrigo, Raoul, Ronald, and Simon. Robin, you thanked me and Nima for welcoming you in our office: I would like to thank you for (irrationally) deciding to try to work in our office, and for making it more lively. Similarly, thanks Anna for dropping by in the last weeks of your PhD. Simon, thanks for your calls and pep talks during the second year of my PhD. Tianwei, thanks for all the chats about PhD life, and life in general. And thanks also for recommending several indies; one can never have enough in their Steam library. Thanks to the Logic Group at the University of Gothenburg for welcoming and hosting me on a few occasions, and especially to Giacomo for our times at conferences and ESSLLI. Marco, thanks for your help on our work with Luca and Claudio. Apostolos, Davide, Giuseppe, Marianna, and Rustam, thanks for agreeing to give VvL Essentials talks; Rustam, thanks also for helping carry Nima safely back to his apartment in Ljubljana. Finally, thanks to Damiano and Johannes for the many video calls, where we worked but sometimes also just hanged out: I had tons of fun, and I hope you too. Damiano, thanks also for hosting me in Tenerife, and for taking care of me and my busted knee.

Thanks to my family and friends for being a safe haven in Italy whenever I come back. Someone I owe big thanks to is Daniele, who got me into moving my body after years of just hating physical activity. Thanks Gabriele for all the afternoons spent playing Magic in the first year of my PhD, and for all the times you visited me. Thanks Daniele, Davide, and Edoardo for having me join your offices to work when in Rome; Daniele and Davide deserve a special mention also for being great study partners throughout my years in Rome. Special thanks are then due to Cristian, Lorenzo (NONNOBELLO), and Lorenzo (Puglia) for the countless gaming afternoons and nights we have had. I have lost count of the games I have bought to play with you, even for just a night, and I would buy them all over again. Thanks for our trips together, for all the times you came to visit me, and for being just so amazing. Thanks to Damiano, Damyán, Edik, Elitsa, Martina, Max, Mila, and Zamma for all the time spent together in Amsterdam. And Max, thanks again for motivating me to move there in the first place. Thanks Stephan for all the help in the gym here in Utrecht, and for the company while training. Finally, thanks to Lise's family for welcoming me: living away from home and loved ones is hard, but thanks to you it is a bit easier.

I saved my last thanks for those who have, in their own ways, given me so much love in the past few years; I hope I can, at least partly, return it in what follows.

Bahareh, you have been first an amazing teacher, and then an even better friend and confidant. We first met during my time at the ILLC, where I took your course and then worked on a project with you. I can confidently say that those were the most formative experiences I had during the MoL. We then met again a few years later, when I was deep in my PhD and not doing the best. You graciously helped me by offering your wisdom, care, and pictures of your adorable cats. Since then, we have been there for each other, and, even when we do not talk or see each other in a while, you still feel close by. I do not think one can ask for a better friend.

Lise, I met you (circa) halfway through my PhD, after a quite rough year. From the first time I saw you, I had the feeling you were someone I could build what we have now with. You then entered my life immediately, and have showed so much to me. You have pushed me to visit places and have experiences that I would have never even thought of, but, most importantly, to be a better person. Thanks for introducing me to so many great people – shout out in particular to Rob, who I can describe as an amazing TV series recommender and just a wonderful *broertje*. Thanks for taking care of me and for trying so hard, every day, to be the amazing and inspiring woman you are, and thanks for letting me be your *weirdootje*. I am not sure I could convey how much you mean to me and how lucky I feel to have you, but know that you have made the last years of my PhD unforgettable, and I will be forever grateful for this.

Infine, grazie mamma e papà. Mi avete supportato in tutto, da quando ero piccolo fino alla mia decisione di trasferirmi nei Paesi Bassi e poi di continuare a vivere là. Il vostro affetto e amore incondizionato mi hanno permesso di essere chi sono oggi. A volte mi chiedete se aveste potuto fare di meglio, spesso col dubbio che effettivamente abbiate sbagliato qualcosa: onestamente, non avete sbagliato nulla, e non sarei dove sono senza voi e ciò che avete fatto per me. Vi voglio bene, e tutto questo è dedicato a voi.