Semantically Labelled Automata for Multi-Task Reinforcement Learning with LTL_f^+ Obligations



Sania Sinha

University of Oxford

A thesis submitted for the degree of

Master of Science

in Mathematics and Foundations of Computer Science

August 2026

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Giuseppe de Giacomo, for his guidance, encouragement, and support throughout the course of this project. His feedback and direction were invaluable in shaping this work and helping me navigate the challenges that arose along the way. I am also very grateful to his students, Mathias, Chris, and Max for their guidance, thoughtful discussions, and help with troubleshooting throughout the project. Their willingness to share their time and expertise made a significant difference to my experience working on this thesis.

I would also like to thank my friends at Oxford for their friendship and support throughout the thesis process. Sharing the challenges, long hours, and inevitable stresses of the timeline made the experience considerably more enjoyable. Finally, I am grateful to my friends, Uzair and Sid, for their suggestions, encouragement, and continued support, which were greatly appreciated throughout the development of this work.

Abstract

Tasks specified as Linear Temporal Logic (LTL) instructions have been successfully adopted into Reinforcement Learning (RL). In this work, we consider a multi-task RL setting where an agent learns a universal policy capable of following arbitrary and unseen tasks. Following the example of the recent task embedding technique using semantically labelled automata [15, 14] devised in [1], we develop the counterpart for tasks expressed in LTLf+ Obligations fragment that supports both finite trace and infinite trace safety and guarantee properties. The devised methodology uses Deterministic Weak Automata (DWA) instead of a Limit-Deterministic Büchi Automata (LDBA) that avoids the complication of non-deterministic choices for the agent present in previous works. Experimental results show that SemLTL$_f^+$ remains faster than and competitive with SemLTL across a range of temporal task families despite its substantially smaller task representation. A guarantee-oriented preliminary model achieves comparable performance on seen guarantee tasks and exhibits zero-shot transfer to previously unseen specification structures, including successful execution of pure safety objectives despite never being explicitly trained on safety formulae. Broader experiments further investigate the effect of extending the training distribution to safety and mixed safety–guarantee obligations. These results demonstrate that deterministic weak automata provide a promising alternative foundation for semantic task representations in temporal MTRL, while identifying representation capacity and training-distribution design as important directions for further investigation.

Contents

1 Introduction **1**

2 Foundations of Sequential Decision Making **6**

2.1 Markov Decision Process (MDP) 7

2.2 Value Functions and Bellman Equations 10

2.3 Reinforcement Learning 11

2.4 Model-Free Reinforcement Learning Algorithms 12

2.4.1 Value-based Methods 13

2.4.2 Policy-based Methods 14

2.4.3 Actor-Critic Methods 15

2.5 Multi-Task Reinforcement Learning 16

2.5.1 Task Representations and Policy Conditioning 17

2.5.2 Generalisation in MTRL and its Challenges 18

3 Linear Temporal Logics **21**

3.1 Linear Temporal Logic 23

3.1.1 Limit-Deterministic Automata and ϵ Transitions 24

3.2 Linear Temporal Logic on Finite Traces 27

3.3 Extending LTL_f to Infinite Traces 29

3.4 LTL_f^+ Obligations 30

3.5 MDPs for Temporal Goals 33

3.5.1 Automata as Finite Task Memory 34

4 Multi-Task Reinforcement Learning with LTL_f^+ Obligations **36**

4.1 Temporal Task Representations in Reinforcement Learning 37

4.1.1 Reward Machines and Structured Reward Representations . . 38

4.1.2 Temporal Logic as a Task Interface 39

4.2 Universal Progress Interface 40

4.2.1 Formula Progression Approach 40

4.2.2 Automata-Theoretic Approach . . . 42
4.2.3 Compositional and Modular Approaches . . . 44
4.3 Towards Semantic Labelled Automata for Representations . . . 44

5 Semantically Labelled Automata 47
5.1 Overview of SemLTL and SemLTL$_f^+$ . . . 48
5.2 Automaton Construction . . . 49
5.2.1 Semantically Labelled LDBAs . . . 49
5.2.2 SemLTL$_f^+$: Obligation-Based DWA Construction . . . 50
5.3 Semantic Labels and Task Embeddings . . . 52
5.3.1 SemLTL$_f^+$: A Single DWA Semantic Label . . . 52
5.3.2 Trueness and its Variations . . . 54
5.3.3 Propositional Attention . . . 56
5.4 RL Pipeline: Removing ϵ-Actions . . . 58
5.4.1 Learnable ϵ-Actions . . . 59
5.4.2 SemLTL$_f^+$: Environment Actions Only . . . 59
5.4.3 Consequences for the Universal Progress Interface . . . 60
5.5 End-to-End SemLTL$_f^+$ Framework . . . 60

6 Experimental Evaluation 63
6.1 Implementation . . . 64
6.1.1 Automaton Construction . . . 64
6.1.2 Reinforcement Learning . . . 65
6.2 Experimental Setup . . . 66
6.2.1 Expressive Equivalence of LTL$_f^+$ and LTL . . . 68
6.2.2 Environment & Learning . . . 72
6.2.3 Curriculum & Task Specification Types . . . 73
6.3 Results . . . 75
6.3.1 Preliminary Results . . . 76
6.3.2 SemLTL$_f^+$ Performance . . . 78

7 Conclusion 82

Bibliography 88

List of Figures

3.1 An example of an LTL trace with a formula satisfied by it. 24
3.2 The Manna-Pnueli Hierarchy. [Taken from lecture material used by Dr. Giuseppe de Giacomo at University of Oxford] 30

5.1 SemLTL Pipeline taken from [1]. 48

Chapter 1

Introduction

A central objective of Artificial Intelligence (AI) is the development of autonomous agents capable of learning, reasoning, and acting effectively across a broad and potentially open-ended range of tasks. Recent advances in foundation models have demonstrated remarkable progress towards this objective in language, vision, and multimodal reasoning. In particular, large-scale models can increasingly interpret previously unseen instructions, reuse knowledge acquired across tasks, and adapt their behaviour without requiring a separately trained model for every new objective. Extending comparable generalisation capabilities to agents operating in sequential and interactive environments, however, remains a fundamental challenge. Reinforcement Learning (RL) provides a natural framework for sequential decision making, in which an agent learns a policy through interaction with an environment. RL methods have achieved strong performance across domains including games, navigation, and robotic control. Nevertheless, the conventional RL formulation typically assumes a single objective encoded by a fixed reward function. The resulting policy is therefore closely coupled to the task for which it was trained. When the objective of the task or the environment changes, e.g. the object to be picked up by a factory robot changes, these policies often require expensive retraining or fine-tuning from scratch.

Multi-Task Reinforcement Learning (MTRL) seeks to overcome this limitation by learning a single policy capable of executing a family of related tasks. Rather than learning an independent policy for every objective, a multi-task agent is conditioned on a representation of the task and learns behaviours that can be shared and composed across specifications. A particularly ambitious objective is *zero-shot task generalisation*: at deployment time, the agent should be capable of interpreting and executing previously unseen combinations of objectives without additional policy optimisation. Such generalisation has motivated a growing body of work on structured task representations, including representations based on temporal logic [42]. Temporal logics

provide a particularly suitable language for specifying the objectives of sequential agents. Unlike scalar rewards or simple goal states, temporal specifications can explicitly describe how desired behaviour should evolve over time. Linear Temporal Logic (LTL) [31], for example, can express sequencing requirements ("visit A before B"), eventual goals ("eventually reach G"), persistent requirements ("eventually remain in S"), and safety constraints ("never enter C"). Since these specifications are symbolic and compositional, they also provide a task representation whose structure is independent of a particular environment or policy architecture.

The use of temporal logic introduces an additional difficulty, however. The desirability of an action may depend on the history of the interaction. The current environment state alone may not indicate which portions of a specification have already been satisfied or which requirements remain outstanding. A common solution is therefore to translate the temporal specification into an automaton whose state tracks progress through the task. The current automaton state can then be combined with the environment state to provide the policy with the memory required to act according to the temporal objective. Automata-theoretic representations have consequently been used extensively for reinforcement learning under temporal-logic specifications [20, 18, 42]. While this approach provides an effective mechanism for tracking progress through an individual specification, the multi-task setting introduces a further representational problem. Automata generated independently for different tasks generally have unrelated state spaces. An automaton state represented only by an index such as q_i carries no intrinsic information about the temporal requirements it represents, nor does it provide a meaningful correspondence with states belonging to other task automata. Consequently, a universal policy conditioned on opaque automaton-state identifiers has limited information with which to recognize that two states from different specifications may encode similar remaining objectives.

Recent work on *Semantically Labelled Automata* addresses this problem for multi-task reinforcement learning with LTL instructions (SemLTL) [1]. Rather than treating automaton states solely as discrete identifiers, SemLTL associates them with semantic information describing the temporal requirements represented by each state. These labels can be embedded into a common numerical representation and supplied to a universal policy, allowing the policy to condition its behaviour on the semantics of the remaining task. This provides a mechanism for sharing information across independently generated task automata and, consequently, for generalizing to specifications not encountered during training. The central insight underlying SemLTL is therefore broader than its particular LTL automata construction: *the automaton*

used to track a temporal task should expose the semantics of its states to the learning agent. This thesis builds upon this principle while considering an alternative temporal specification language and automata-theoretic pipeline.

Specifically, we investigate multi-task reinforcement learning with the *obligation fragment of* LTL_f^+. This fragment captures positive Boolean combinations of safety and guarantee properties while admitting representations using Deterministic Weak Automata (DWAs). These properties make LTL_f^+ obligations an attractive setting in which to investigate semantically labelled task representations: the fragment retains temporally extended objectives relevant to sequential control while providing a deterministic automata-theoretic foundation for tracking their satisfaction. Accordingly, this thesis develops an LTL_f^+ obligation counterpart to SemLTL. We replace the automata pipeline used for general LTL specifications with a DWA-based construction for LTL_f^+ obligations, while retaining the central idea of exposing the semantic meaning of automaton states to a universal reinforcement learning policy. The resulting framework separates the symbolic mechanism used to track temporal progress from the numerical representation used by the policy: the automaton determines the current task state, while semantic labels associated with that state communicate the remaining temporal requirements to the learning agent. This leads to the central question investigated in this thesis:

> *Can semantically labelled deterministic weak automata provide an effective and computationally desirable task representation for multi-task reinforcement learning with* LTL_f^+ *obligation specifications?*

We investigate this question through three subsidiary research questions.

RQ1. How can the semantic-labelling paradigm introduced for general LTL specifications be adapted to LTL_f^+ obligations represented using Deterministic Weak Automata?

RQ2. Can semantic representations derived from these automata support a universal policy that transfers across temporal specifications, including specifications not encountered during training?

RQ3. What computational and learning trade-offs arise from replacing the general-LTL automata pipeline with the deterministic construction available for LTL_f^+ obligations?

The principal contributions of this thesis are as follows.

- **Semantic automata for LTL_f^+ obligations.** We extend the semantically labelled automata paradigm to the LTL_f^+ obligation fragment, constructing task representations that expose the semantic requirements associated with automaton states rather than treating those states as opaque identifiers.
- **Deterministic task progression using DWAs.** We employ Deterministic Weak Automata as the underlying representation for LTL_f^+ obligation specifications. This provides deterministic progression through temporal tasks and removes the need for the reinforcement learning agent to resolve automaton nondeterminism as part of its policy.
- **Computational and empirical evaluation.** We evaluate the proposed framework against relevant LTL-based multi-task reinforcement learning approaches, examining both its computational characteristics and its ability to learn and generalize across temporally extended tasks.

Together, these contributions investigate whether the semantic-automata approach to multi-task reinforcement learning can be realised using a more specialised deterministic temporal-logic pipeline. In doing so, this thesis explores the trade-off between the generality of the specification language, the computational properties of its automata representation, and the semantic information required for generalization across tasks.

Thesis Outline

This thesis is organised into five chapters, progressing from the foundations of sequential decision making and temporal specification to the proposed semantically labelled automata framework and its empirical evaluation. **Chapter 2** introduces the foundations of sequential decision making, covering Markov Decision Processes, reinforcement learning, and Multi-Task Reinforcement Learning (MTRL). **Chapter 3** presents the temporal-logic foundations of this work, progressing from LTL and LTL_f to LTL_f^+ obligations and their use in specifying temporal goals for sequential decision-making problems. **Chapter 4** considers Multi-Task Reinforcement Learning with LTL_f^+ obligations and introduces Universal Progress Interfaces (UPIs), comparing formula-progression and automata-theoretic approaches to representing progress through temporal tasks. **Chapter 5** presents the main methodological contribution of this thesis. Building on SemLTL, it develops semantically labelled automata for

LTL_f^+ obligations and introduces semantic representations of automaton states for multi-task policy conditioning. **Chapter 6** describes the implementation and experimental setup used to evaluate the proposed framework, presents the resulting empirical findings, and discusses their implications, limitations, and directions for future work.

Chapter 2

Foundations of Sequential Decision Making

This chapter establishes the foundations of sequential decision making required throughout this thesis. We begin with Markov Decision Processes (MDPs), which provide the mathematical framework underlying reinforcement learning, and introduce policies, returns, value functions, and the Bellman equations. We then consider the reinforcement learning setting in which the environment dynamics are not assumed to be known, focusing on the model-free methods relevant to the experimental methodology of this work. Finally, we extend the single-task formulation to Multi-Task Reinforcement Learning (MTRL), with particular emphasis on task-conditioned policies, shared representations, and generalisation across objectives. These concepts establish two requirements that become central in subsequent chapters. First, the information supplied to a reinforcement learning policy must provide a sufficient representation of the decision process: when the current environment state does not capture information relevant to future rewards, the standard Markovian formulation is no longer directly applicable. Second, a multi-task policy requires a representation of the current objective that exposes sufficient common structure for knowledge to transfer across tasks. The following chapters address these requirements for temporally extended objectives by introducing temporal-logic specifications and structured representations of task progress.

The standard formal framework for modeling decision-making under uncertainty is the Markov Decision Process (MDP) [32, 39, 2]. The precursor to the modern MDP originated in 1906 with Russian mathematician Andrey Andreyevich Markov, who introduced Markov Chains to prove that statistical laws of large numbers could apply to dependent random variables. Although Markov chains established the core memoryless property, where future states depend solely on the present state rather

than historical trajectories, they were purely descriptive stochastic models that lacked any mechanism for active decision-making, control choices, or performance rewards. The modern Markov Decision Process (MDP) was formally synthesized in the 1950s by Richard Bellman. In his landmark 1957 work *Dynamic Programming* [6], Bellman introduced the Principle of Optimality and the recursive Bellman Equations. By combining states, actions, transition probabilities, and discounted future rewards into a unified framework, Bellman showed how complex multi-stage decision problems could be decomposed into solvable single-step recursive updates. While classical dynamic programming offered exact solutions, it required complete knowledge of environment transition dynamics and reward functions. To bypass this limitation, Chris Watkins introduced Q-learning [46] in 1989, proving that agents could learn optimal Bellman value functions directly from environment interaction without a prior model. Richard Sutton and Andrew Barto subsequently consolidated this sample-based paradigm in 1998, establishing the MDP as the universal mathematical formulation for reinforcement learning.

In recent years, the standard MDP framework has been extended through neuro-symbolic paradigms to handle complex tasks where goals depend on the history of execution rather than just immediate state transitions [19, 1]. These extensions and their integration with formal temporal logic instructions are discussed in detail later in Chapter 3.

2.1 Markov Decision Process (MDP)

An MDP represents an environment where an agent interacts sequentially at discrete time steps $t \in \{0, 1, 2, \dots\}$. Formally, we can define it as follows.

Definition 2.1 (MDP)**.** A Markov Decision Process is defined as a tuple,

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma, \mu_0)$$

where

- $\mathcal{S}$ is the set of environment states (state space).
- $\mathcal{A}$ is the set of allowable actions (action space).
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Pr(\mathcal{S})$ is the transition probability function, defining $\mathcal{P}(s' \mid s, a) = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, specifying the scalar immediate reward on taking an action, $R(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$.
- $\gamma \in [0, 1)$ is the discount factor prioritizing immediate over future rewards.
- $\mu_0 : \mathcal{S} \to [0, 1]$ is the initial state distribution.

The core underlying assumption of an MDP is the **Markov Property** where the future state S_{t+1} depends only on the current state S_t and action A_t, conditionally independent of all preceding states and actions. Anything having this property is said to be *Markovian*. Mathematically, let the complete history at timestep t be defined as follows.

$$H_t = (S_0, A_0, ..., S_{t-1}, A_{t-1}, S_t) \tag{2.1}$$

Then the Markov Property equation is:

$$\Pr(S_{t+1} = s' \mid H_t, A_t = a) = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a) \tag{2.2}$$

A policy specifies the behaviour of an agent within an MDP. In the most general stochastic case, a policy assigns a probability distribution over actions for every state.

Definition 2.2 (Policy)**.** A stochastic policy is a mapping

$$\pi : \mathcal{S} \to \Delta(\mathcal{A}),$$

where $\Delta(\mathcal{A})$ denotes the space of probability distributions over the action space $\mathcal{A}$. Equivalently,

$$\pi(a \mid s) = \Pr(A_t = a \mid S_t = s)$$

denotes the probability of selecting action a when the agent is in state s.

A deterministic policy is a special case in which a single action is selected for each state, $\pi : \mathcal{S} \to \mathcal{A}$.

Given an initial-state distribution μ_0, a policy π, and the transition dynamics of the MDP, interaction between the agent and the environment induces a distribution over trajectories.

Definition 2.3 (Trajectory)**.** A trajectory is a sequence of states, actions, and rewards generated by repeated interaction between the agent and the environment:

$$\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots).$$

For a finite-horizon problem with horizon T, a trajectory may be written as

$$\tau = (S_0, A_0, R_1, S_1, \ldots, A_{T-1}, R_T, S_T).$$

Assuming that rewards are determined by the transition or are represented separately through the reward function, the probability of a finite state-action trajectory under policy π is as follows.

$$p_\pi(\tau) = \mu_0(s_0) \prod_{t=0}^{T-1} \pi(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t) \tag{2.3}$$

This factorisation follows directly from the Markov property: the probability of each successor state depends only on the current state and action, while the probability of the selected action is determined by the policy. The quality of a trajectory is measured through its cumulative discounted reward.

Definition 2.4 (Return)**.** The discounted return from timestep t is

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where $\gamma \in [0, 1)$ is the discount factor.

For a finite-horizon problem with terminal time T, this becomes $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$. The discount factor determines the relative importance of immediate and delayed rewards. Values of γ close to zero favour short-term rewards, whereas values closer to one assign greater importance to long-term consequences. The objective of reinforcement learning is to find a policy that maximises the expected return induced by its trajectories.

Definition 2.5 (Policy Objective)**.** For a finite-horizon MDP, the expected performance of a policy π is defined as

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi} \left[\sum_{t=0}^{T-1} \gamma^t R_{t+1} \right].$$

Equivalently, when considering the return from the initial timestep, $J(\pi) = \mathbb{E}_{\tau \sim p_\pi} [G_0]$. The expectation is taken over all sources of stochasticity in the interaction, including the initial-state distribution, the environment transition dynamics, and the stochastic policy.

Definition 2.6 (Optimal Policy)**.** An optimal policy is any policy that maximises the expected return. That is,

$$\pi^* \in \arg\max_\pi J(\pi).$$

More locally, an optimal policy can be characterised through the optimal action-value function Q^* as $\pi^*(s) \in \arg\max_{a \in \mathcal{A}} Q^*(s, a)$. This trajectory-based formulation is particularly useful for policy-gradient methods, where the objective is differentiated directly with respect to the parameters of a policy π_θ. It also provides a natural foundation for the multi-task setting considered later, where both the trajectory distribution and the policy are conditioned on an additional task representation.

2.2 Value Functions and Bellman Equations

To evaluate policy quality, RL relies on value functions, which quantify the long-term expected return from a given state or state-action pair [40].

Definition 2.7 (State-Value and Action-Value Functions)**.** For a policy π, the state-value function $V^\pi(s)$ and the action-value function $Q^\pi(s, a)$ are defined as follows.

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Using the Markov property, value functions satisfy fundamental recursive relations known as the **Bellman Expectation Equations**.

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^\pi(s') \right] \end{aligned}$$

Similarly, we have

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) \left[\mathcal{R}(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^\pi(s', a') \right]$$

where s' represents the next state that the environment transitions into at time step $t + 1$, after the agent takes action a in current state s at time step t as defined in Definition 2.1.

The central objective in RL is to find an optimal policy π^* that achieves the maximum possible expected return in all states. A policy π is defined as better than

or equal to π' if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$. All optimal policies share the same unique optimal state-value function $V^*(s)$ and optimal action-value function $Q^*(s, a)$.

$$V^*(s) \doteq \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}$$

$$Q^*(s, a) \doteq \max_{\pi} Q^{\pi}(s, a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

The relationship between $V^*(s)$ and $Q^*(s, a)$ is straightforward.

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a)$$

Because optimal policies assign all probability mass to action(s) maximizing the expected return, the Bellman expectation equations collapse into non-linear system equations called the **Bellman Optimality Equations**.

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) \left[\mathcal{R}(s, a, s') + \gamma V^*(s')\right] \tag{2.4}$$

$$Q^*(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) \left[\mathcal{R}(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')\right] \tag{2.5}$$

Once $Q^*(s, a)$ is determined, an optimal policy π^* can be obtained deterministically by taking the greedy action in any state as follows.

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$$

2.3 Reinforcement Learning

A Markov Decision Process specifies the structure of a sequential decision-making problem: it defines the state and action spaces, the transition dynamics of the environment, the reward function, and the criterion according to which behaviour is evaluated. Reinforcement Learning (RL), in contrast, concerns how an agent can *learn* an effective policy for such a decision problem through interaction with the environment [39]. In the standard reinforcement learning setting, the agent is not assumed to know the optimal policy in advance and typically does not have direct access to the environment transition probabilities or reward model. Instead, it collects experience by interacting with the environment and uses this experience to improve its behaviour over time. The objective remains to find a policy

$$\pi^* \in \arg\max_{\pi} J(\pi),$$

but the policy must now be inferred from sampled trajectories rather than computed directly from a completely known model of the environment.

A central distinction in reinforcement learning lies between *model-based* and *model-free* approaches. While model-based methods require or learn an explicit model of the environment's transition dynamics $\mathcal{P}(s' \mid s, a)$ and reward distribution $\mathcal{R}$, model-free RL bypasses the need to explicitly model $\mathcal{P}$ and $\mathcal{R}$. Instead, model-free algorithms optimize policies or value functions directly from sampled environment trajectories (s, a, r, s').

The remainder of this chapter focuses primarily on model-free reinforcement learning, since these methods form the basis of the learning algorithms used later in this thesis. Model-free approaches can themselves be divided into three broad families according to the object that is learned.

- **Value-based methods** estimate value functions such as $V^\pi(s)$ or $Q^\pi(s, a)$ and derive a policy implicitly by selecting actions that maximise the estimated value. Q-learning and Deep Q-Networks are representative examples.
- **Policy-based methods** directly parameterise the policy $\pi_\theta(a \mid s)$ and optimise its parameters with respect to the expected return $J(\theta)$. Policy-gradient methods belong to this family.
- **Actor-critic methods** combine the previous two approaches. An *actor* represents and updates the policy, while a *critic* estimates a value function that provides a learning signal for the actor. Algorithms such as PPO and SAC are commonly formulated within this framework.

These categories are not always mutually exclusive in practice. In particular, many modern deep reinforcement learning algorithms combine elements from multiple families. Nevertheless, the distinction provides a useful conceptual framework for introducing the learning methods considered in the remainder of this chapter. The following sections first examine value-based learning through Monte Carlo and temporal-difference methods, before considering direct policy optimisation and actor-critic approaches.

2.4 Model-Free Reinforcement Learning Algorithms

Having established model-free RL as the paradigm for optimizing policies from sampled experience tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ without explicit knowledge of $\mathcal{P}$ or $\mathcal{R}$, the

central computational question becomes *how* an agent utilizes these samples to approximate the optimal value functions (Q^*, V^*) or policy (π^*). Algorithmic solutions to this problem generally fall along a spectrum determined by their optimization mechanics and can be separated into three primary categories: value-based, policy-based, and actor-critic methods.

2.4.1 Value-based Methods

Value-based methods learn an approximation of the optimal action-value function $Q^*(s, a)$ and infer an optimal policy implicitly by selecting actions greedily using $\pi(s) = \arg\max_a Q(s, a)$.

Definition 2.8 (Q-Learning)**.** Q-Learning (Tabular) [46] is an off-policy Temporal-Difference (TD) control algorithm that updates state-action values using the TD error δ_t as follows.

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

where

- $\alpha \in (0, 1]$ is the learning rate parameter,
- $r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a')$ is the Temporal-Difference (TD) Target,
- $\delta_t \doteq r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)$ is the TD Error.

Under standard Robbins-Monro conditions on the learning rate sequence α_t, $Q(s, a)$ is guaranteed to converge asymptotically to the optimal action-value function $Q^*(s, a)$ for finite MDPs.

When state spaces are continuous or excessively large, tabular Q-learning becomes intractable. To scale Q-learning, Deep Q-Networks (DQN) [28] replace the lookup table with a deep neural network parameterized by weights θ. The network acts as a function approximator, learning to predict the optimal action-values $Q^*(s, a; \theta)$ directly from raw state observations. Standard supervised learning assumes that training data is independent and identically distributed (i.i.d.), but an RL agent collecting sequential data violates this assumption: consecutive states are highly correlated, and the target values change constantly as the network updates. To overcome this instability, DQN introduces two key architectural innovations.

1. Experience Replay Memory ($\mathcal{D}$): Transitions (s, a, r, s') are stored in a buffer and randomly sampled during mini-batch updates to break temporal correlations between sequential steps.

2. Target Network (θ^-): A separate set of parameters θ^-, periodically updated to track θ, is used to calculate the target value:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

2.4.2 Policy-based Methods

Value-based methods struggle in continuous or high-dimensional action spaces, as computing $\max_{a\in\mathcal{A}} Q(s, a)$ becomes computationally prohibitive. Policy Gradient methods parameterize the policy explicitly as $\pi_\theta(a \mid s)$ with weights θ and optimize performance directly via gradient ascent on the expected return $J(\theta) \doteq \mathbb{E}_{\pi_\theta}[G_0]$.

Theorem 2.4.1 (Policy Gradient Theorem [40])**.** *Let $J(\theta) \doteq \mathbb{E}_{\pi_\theta}[G_0]$ define the performance measure of a parameterized policy $\pi_\theta(a \mid s)$ with parameters $\theta \in \mathbb{R}^d$. For any differentiable policy π_θ and any episodic or continuing Markov Decision Process with stationary dynamics, the gradient of the performance objective with respect to the policy parameters is given by*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^{\pi_\theta}(s, a)\right] \tag{2.6}$$

or equivalently, expressed across the stationary state distribution $d^{\pi_\theta}(s)$ as follows.

$$\nabla_\theta J(\theta) = \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) Q^{\pi_\theta}(s, a) \tag{2.7}$$

where $d^{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid s_0, \pi_\theta)$ represents the discounted state visitation distribution under policy π_θ.

Thus, according to the Policy Gradient Theorem, the gradient of the performance objective with respect to policy parameters θ does not depend on the derivative of the unknown environment transition dynamics. Replacing $Q^{\pi_\theta}(s, a)$ with the empirical return G_t yields the baseline REINFORCE algorithm. To reduce the variance associated with Monte Carlo returns, a state-value baseline $V(s)$ is typically subtracted, yielding the **Advantage Function** as follows.

$$A^{\pi_\theta}(s, a) \doteq Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$$

2.4.3 Actor-Critic Methods

While pure value-based methods learn action values and infer a policy, and pure policy gradient methods optimize actions directly, Actor-Critic frameworks [23] unify both paradigms. The Actor updates the policy parameters θ to select actions, while the Critic updates value function parameters ϕ (such as $V_\phi(s)$ or $Q_\phi(s,a)$) to evaluate how good those actions were. This architecture substantially reduces variance compared to raw Monte Carlo return estimates.

Proximal Policy Optimization (PPO). PPO [36] is an on-policy actor-critic algorithm that mitigates destructive policy updates (a common failure mode in standard policy gradients) without the computational overhead of Trust Region Policy Optimization (TRPO). It introduces a clipped surrogate objective as follows.

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

where

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ represents the probability ratio between the new and old policy,
- $\hat{A}_t$ is the estimated Advantage,
- ϵ bounds the step size.

It compares how much the new policy differs from the old policy. If the proposed update tries to change the action probabilities too drastically for a given advantage estimate, PPO simply ignores the extra incentive, preventing destructive policy collapses while remaining computationally lightweight (unlike second-order trust-region methods).

Soft Actor-Critic (SAC). SAC [17] is an off-policy actor-critic algorithm designed for continuous control. It operates within the Maximum Entropy RL framework, expanding the standard reward objective to maximize both expected return and policy entropy, $\mathcal{H}(\pi_\theta(\cdot \mid s_t))$.

$$J(\theta) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[R(s_t, a_t, s_{t+1}) + \alpha\, \mathcal{H}\left(\pi_\theta(\cdot \mid s_t)\right) \right]$$

where α is a temperature parameter balancing exploration (entropy maximization) and exploitation (reward optimization).

The agent must maximize its expected cumulative reward while simultaneously maximizing the entropy (randomness) of its policy. This prevents early collapse to

a single sub-optimal policy, encourages continuous exploration of promising action paths, and produces policies that are remarkably robust to environmental perturbations. SAC achieves superior sample efficiency and robustness against multi-modal local optima, making it a benchmark framework in continuous task settings.

2.5 Multi-Task Reinforcement Learning

The reinforcement learning formulation introduced thus far considers the optimisation of a policy for a single decision-making problem. In many applications, however, an agent may be required to perform a collection of related tasks within the same or similar environments. Training an independent policy for every task can be computationally expensive and prevents the agent from exploiting behavioural structure shared across tasks. Multi-Task Reinforcement Learning (MTRL) instead seeks to learn a shared policy capable of solving multiple tasks while transferring knowledge between them [8, 48, 44, 41].

Formulation

Let $\mathcal{T}$ denote a task drawn from a distribution of tasks $p(\mathcal{T})$. A task may, in general, induce a different MDP, $\mathcal{M}_{\mathcal{T}} = (\mathcal{S}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}}, P_{\mathcal{T}}, R_{\mathcal{T}}, \gamma_{\mathcal{T}}, \mu_{0,\mathcal{T}})$. Multi-task learning is particularly natural when the tasks share substantial structure. In the setting considered throughout this thesis, tasks are assumed to operate within a common environment and therefore share the state space, action space, and transition dynamics, while differing primarily in the objective that the agent is required to satisfy. Such a family can be written as $\mathcal{M}_{\mathcal{T}} = (\mathcal{S}, \mathcal{A}, P, R_{\mathcal{T}}, \gamma, \mu_0)\,, \qquad \mathcal{T} \sim p(\mathcal{T})$.

Rather than learning a separate policy $\pi_{\mathcal{T}}$ for each task, MTRL aims to learn a single parameterised policy whose behaviour can be adapted according to the current task. Let $z_{\mathcal{T}} = f(\mathcal{T}) \in \mathcal{Z}$ denote a representation of task $\mathcal{T}$. A task-conditioned policy is then defined as

$$\pi_\theta : \mathcal{S} \times \mathcal{Z} \to \Delta(\mathcal{A}), \tag{2.8}$$

with $\pi_\theta(a \mid s, z_{\mathcal{T}})$ representing the probability of selecting action a in state s while pursuing task $\mathcal{T}$. The corresponding multi-task objective is to maximise expected performance over the task distribution.

$$J(\theta) = \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[J_{\mathcal{T}}(\pi_\theta) \right] \tag{2.9}$$

Equivalently, this can be expressed in terms of trajectories as follows.

$$J(\theta) = \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}\left[\mathbb{E}_{\tau\sim p_{\pi_\theta}(\tau|z_\mathcal{T})}\left[\sum_{t=0}^{T-1}\gamma^t R^{\mathcal{T}}_{t+1}\right]\right] \tag{2.10}$$

The shared parameters θ allow experience obtained while solving one task to influence the behaviour learned for other tasks. When tasks contain common structure, such parameter sharing can improve learning efficiency and enable the reuse of behaviours across objectives. However, successful sharing depends critically on the information supplied through the task representation $z_\mathcal{T}$.

2.5.1 Task Representations and Policy Conditioning

The task representation provides the policy with information describing the objective currently being pursued. Its form can vary considerably depending on the structure available in the task family. Common representations include a discrete task identifier, an explicit goal or target state, a structured symbolic specification, or a learned task or context embedding. A discrete task identifier is sufficient to distinguish a fixed collection of training tasks, but provides little information about relationships between them. For example, two tasks may be represented by unrelated one-hot vectors despite requiring highly similar behaviours. Such representations can therefore support parameter sharing without necessarily providing a useful basis for generalisation to new tasks.

When tasks are naturally described by goals, the goal itself can instead be supplied to the policy. Goal-conditioned reinforcement learning augments the state representation with a goal g, yielding a policy of the form $\pi_\theta(a \mid s, g)$. Similarly, Universal Value Function Approximators (UVFAs) extend conventional value functions by conditioning their predictions on the goal [35]:

$$V^\pi(s, g) = \mathbb{E}_\pi\left[G_t \mid S_t = s,\, G = g\right],$$

$$Q^\pi(s, a, g) = \mathbb{E}_\pi\left[G_t \mid S_t = s,\, A_t = a,\, G = g\right].$$

This formulation allows the value function and policy to exploit relationships between different goals rather than associating each objective with an independent set of parameters. More generally, a task description may first be mapped into a learned representation, $z_\mathcal{T} = f_\phi(\mathcal{T})$, after which the policy is conditioned on the resulting embedding, $\pi_\theta(a \mid s, z_\mathcal{T})$. Importantly, goal conditioning and learned task embeddings are

not mutually exclusive approaches. A goal, symbolic instruction, or other structured task description may itself be processed by an encoder f_ϕ to obtain a representation suitable for policy conditioning. The central design question is therefore not simply whether the policy is "goal-conditioned" or "embedding-conditioned", but rather *which properties of the task are preserved by its representation.* This distinction is particularly important when generalisation is desired. A representation that exposes common structure between tasks may allow the policy to reuse behaviours learned for one objective when encountering another, whereas an opaque representation may provide no indication that two tasks are related.

2.5.2 Generalisation in MTRL and its Challenges

Performance across multiple training tasks should be distinguished from the ability to generalise beyond those tasks. Let $\mathcal{T}_{\text{train}} = \{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$ denote the set of tasks encountered during training. A multi-task policy is optimised jointly over these tasks, for example according to $J_{\text{train}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} J_{\mathcal{T}_i}(\pi_\theta)$. Successful performance on $\mathcal{T}_{\text{train}}$ demonstrates that a single policy can solve multiple objectives, but does not by itself establish that the policy has learned a representation that generalises beyond the training distribution. To evaluate generalisation, consider a set of held-out tasks $\mathcal{T}_{\text{test}} = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_M\}$, such that $\mathcal{T}_{\text{train}} \cap \mathcal{T}_{\text{test}} = \varnothing$. We distinguish the following related concepts.

Multi-task learning. A single policy is trained jointly to perform multiple tasks, typically exploiting shared parameters or representations across them.

Transfer. Knowledge acquired while learning one or more tasks improves learning or performance on another task. Transfer may involve additional training on the target task.

Zero-shot task generalisation. A trained policy is evaluated on a previously unseen task without any additional parameter updates. The policy must therefore interpret the new task through the structure encoded in its task representation.

Zero-shot generalisation is a particularly demanding objective. If an unseen task is represented only by an arbitrary identifier, the policy has no principled means of relating that identifier to previously learned behaviour. By contrast, structured representations may expose similarities between training and test objectives and thereby provide a basis for compositional or semantic generalisation. The distinction between

training-task performance and zero-shot performance is central to the setting considered in this thesis. The objective is not merely to train a single policy that memorises a finite collection of temporal tasks, but to provide the policy with a representation through which behaviour can be reused across previously unseen task specifications.

Although parameter sharing can improve data efficiency and facilitate transfer, jointly learning multiple tasks introduces several challenges. In particular, improvements for one task need not improve performance on another. Let $g_i = \nabla_\theta J_{\mathcal{T}_i}(\theta)$ denote the policy gradient associated with task $\mathcal{T}_i$. For two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$, the condition $\langle g_i, g_j \rangle < 0$ indicates that their local gradient directions conflict: an update that improves one task may locally oppose the update favoured by the other. Such gradient conflict is one mechanism through which interference between tasks can arise, although it should not be regarded as a complete definition of negative transfer. More generally, MTRL systems must contend with several forms of interference. Tasks may require incompatible features from a shared representation, may differ substantially in difficulty or reward scale, or may be sampled with different frequencies during training. Consequently, a shared policy does not automatically guarantee beneficial transfer; the task representation and learning procedure must enable useful common structure to be exploited while limiting destructive interference.

A further challenge concerns the amount of task information required by the policy. For conventional goal-conditioned problems, a static task representation $z_\mathcal{T}$ may be sufficient: the objective remains fixed throughout the episode, and the current environment state captures the information necessary to determine future behaviour. This assumption becomes more problematic for temporally extended objectives. Consider, for example, the instruction "visit A before reaching B". Two trajectories may arrive at the same environment state s_t while differing in whether A has previously been visited. Although both trajectories share the same current state and overall task specification, the requirements that remain to be satisfied are different. Thus, $(s_t, z_\mathcal{T})$ need not provide a sufficient representation of the decision problem when the objective depends on the history of the trajectory. This motivates an important distinction between *task identity* and *task progress*. A task representation describes the overall objective being pursued, whereas a progress representation describes the portion of that objective that remains relevant at the current timestep. For temporally extended tasks, the latter must evolve as the agent interacts with the environment. The policy considered later in this thesis will therefore take the more general form

$\pi_\theta(a_t \mid s_t, z_t)$, where z_t is a dynamic representation of progress through the current task rather than merely a static task identifier.

The following chapter introduces temporal logic as a formal language for specifying such temporally extended objectives and develops the machinery required to represent their evolution over trajectories.

Chapter 3

Linear Temporal Logics

While the previous chapter established the core mechanics of Multi-Task Reinforcement Learning, modern goal-conditioned policies remain intrinsically tied to the Markov assumption. Standard scalar reward functions $\mathcal{R}(s_t, a_t, s_{t+1})$ and goal vectors $g \in \mathcal{G}$ evaluate immediate transition quality or terminal geometric proximity. However, real-world robotic and autonomous systems are rarely tasked with simple reachability. They operate under **temporally extended** directives involving sequential dependencies such as "pick up component A, inspect it at station B, and deliver it to C", conditional constraints such as "keep key K held until door D opens", and non-negotiable safety invariants such as "never enter hazard zone H". Notice that such tasks are non-Markovian in nature as the memoryless assumption no longer works. This chapter resolves these issues by integrating formal methods with sequential decision-making. We introduce Linear Temporal Logics as an unambiguous, formal language to specify task directives capable of representing non-Markovian objectives.

To understand why traditional specification methods fall short for these tasks, one must look to the origins of formal system verification. In the mid-twentieth century, as digital computing transitioned from isolated batch-processing algorithms to concurrent, interactive software systems, classical methods of proving program correctness encountered a fundamental barrier. Early verification frameworks, most notably Hoare logic, were designed to reason about transformational programs, which are systems that take an input, perform a computation, terminate, and return an output. For these systems, correctness was fully captured by pre-conditions and post-conditions. However, autonomous agents, operating systems, and industrial controllers belong to an entirely different class of computational entities known as reactive systems. First categorized by David Harel and Amir Pnueli, reactive systems do not terminate; rather, they maintain an ongoing, continuous interaction with their dynamic

environment. For an autonomous agent navigating a warehouse, there is no single post-condition that signals absolute completion. Instead, correctness is continuously evaluated over an infinite execution sequence. The agent must balance immediate reactivity with long-term behavioral requirements, ensuring that safety invariants are never violated while progress toward active goals is continually sustained.

This conceptual shift necessitated a mathematical language capable of expressing properties over sequences of execution states across time. In 1977, Amir Pnueli introduced Linear Temporal Logic (LTL) [31] to computer science, adapting modal logic, originally developed by philosophers such as Arthur Prior to reason about the truth of propositions relative to time, into a formal framework for system specification. Pnueli's critical insight was that time could be modeled as a linear sequence of states, allowing system behaviors to be specified through temporal operators such as $\Box$ ("always"), $\diamond$ ("eventually"), and $\mathbf{U}$ ("until"). LTL transformed formal verification by enabling computer scientists to write declarative, mathematically rigorous specifications for complex temporal behaviors without hard-coding ad-hoc, brittle state-machine logic into the system's runtime code. The subsequent decades saw the rapid expansion of temporal logics from static model checking (verifying whether a known system satisfies a formula) to reactive synthesis, pioneered by Pnueli and Rosner in 1989. While theoretical computer science established the foundational compilation pipelines comprising translating temporal formulas into finite automata via automata-theoretic approaches pioneered by Vardi, Wolper, and Chandra, these classical synthesis techniques relied on explicit knowledge of the environment's state transition graph. In modern autonomous systems, however, explicit environment models are rarely available. The agent must learn to satisfy temporal specifications in continuous, high-dimensional, and uncertain physical spaces where transition dynamics can only be sampled through interaction.

This chapter introduces the temporal-logic foundations required throughout the remainder of this thesis. We begin with classical LTL over infinite traces before introducing Linear Temporal Logic on Finite Traces (LTL_f), which interprets analogous temporal operators over finite executions [12]. We then introduce LTL_f^+, which uses finite-trace LTL_f properties as building blocks for specifying properties over infinite traces [3]. Particular attention is given to the *obligation fragment*, consisting of positive Boolean combinations of safety and guarantee properties.

3.1 Linear Temporal Logic

Linear Temporal Logic (LTL) is a modal temporal logic for specifying properties of infinite sequences of states [31]. Rather than describing a single state in isolation, an LTL formula characterizes how propositions should evolve throughout an execution.

Let $\mathcal{AP}$ denote a finite set of atomic propositions. An *infinite trace* over $\mathcal{AP}$ is an infinite word $\sigma = \sigma_0\sigma_1\sigma_2\cdots \in (2^{\mathcal{AP}})^\omega$, where each $\sigma_i \subseteq \mathcal{AP}$ contains exactly those propositions that are true at timestep i.

Definition 3.1 (LTL Syntax)**.** Given a set of atomic propositions $\mathcal{AP}$, LTL formulas are generated according to the following grammar.

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2,$$

where $p \in \mathcal{AP}$, $\mathbf{X}$ denotes the *next* operator, and $\mathbf{U}$ denotes the *until* operator.

Other Boolean connectives are defined in the usual manner. Two frequently used temporal operators are derived from $\mathbf{U}$:

- $\mathbf{F}\varphi \equiv \top \mathbf{U} \varphi$
- $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$

Here, $\mathbf{F}\varphi$ states that φ must hold at some future position, whereas $\mathbf{G}\varphi$ states that φ must hold at every position.

Definition 3.2 (LTL Semantics)**.** Let $\sigma = \sigma_0\sigma_1\sigma_2\cdots \in (2^{\mathcal{AP}})^\omega$. The satisfaction relation $(\sigma, i) \models \varphi$ is defined inductively as follows.

$$(\sigma, i) \models p \iff p \in \sigma_i, \tag{3.1}$$
$$(\sigma, i) \models \neg\varphi \iff (\sigma, i) \not\models \varphi, \tag{3.2}$$
$$(\sigma, i) \models \varphi_1 \wedge \varphi_2 \iff (\sigma, i) \models \varphi_1 \text{ and } (\sigma, i) \models \varphi_2, \tag{3.3}$$
$$(\sigma, i) \models \mathbf{X}\varphi \iff (\sigma, i+1) \models \varphi, \tag{3.4}$$
$$(\sigma, i) \models \varphi_1 \mathbf{U} \varphi_2 \iff \exists j \geq i : (\sigma, j) \models \varphi_2 \text{ and } \forall k \in [i, j), (\sigma, k) \models \varphi_1. \tag{3.5}$$

An infinite trace σ satisfies φ, written $\sigma \models \varphi$, whenever $(\sigma, 0) \models \varphi$.

For example, the formula $\mathbf{G}\,(\mathsf{request} \rightarrow \mathbf{F}\,\mathsf{response})$ states that every request must eventually be followed by a response. Similarly, $\mathbf{G}\neg\mathsf{hazard}$ specifies a safety requirement that the proposition hazard must never become true. Figure 3.1 depicts a visual example of a formula and an accepting trace.

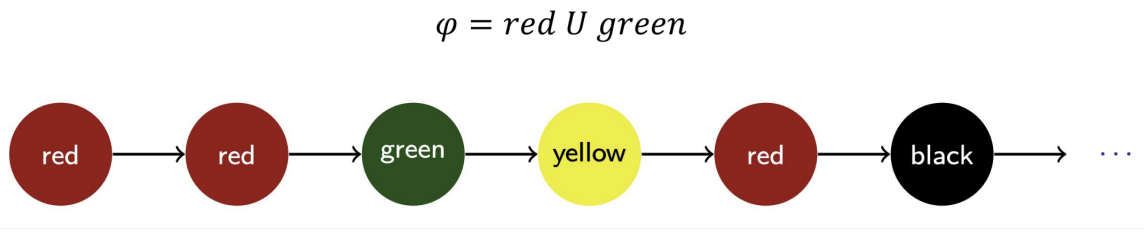


Figure 3.1: An example of an LTL trace with a formula satisfied by it.

3.1.1 Limit-Deterministic Automata and ϵ Transitions

A fundamental connection between temporal logic and automata theory is that temporal specifications can be represented by automata operating on the same traces over which the formulas are interpreted. In the classical automata-theoretic approach to verification, an LTL formula φ is translated into an automaton $\mathcal{A}_\varphi$ whose accepted ω-language corresponds to the traces satisfying φ [43]:

$$\mathcal{L}(\mathcal{A}_\varphi) = \{\sigma \in (2^{\mathcal{AP}})^\omega \mid \sigma \models \varphi\}.$$

For infinite-trace LTL, this requires an automaton equipped with an ω-acceptance condition, such as a Büchi [38], Rabin [33], parity [9, 13], or related acceptance condition. The automaton reads one propositional valuation $\sigma_t \in 2^{\mathcal{AP}}$ at each timestep and updates its internal state accordingly. This automata-theoretic interpretation is particularly important for sequential decision making. Whereas the temporal formula describes an objective declaratively, the automaton provides an operational representation of progress through that objective. Its state can therefore serve as a finite-memory summary of the portion of the observed trace that is relevant to future satisfaction.

For general LTL specifications, fully deterministic automata can be costly to construct and may require richer acceptance conditions. Consequently, many reinforcement-learning approaches instead employ *Limit-Deterministic Büchi Automata* (LDBAs) [37, 18]. An LDBA is a Büchi automaton whose nondeterminism is restricted to a limited portion of the automaton. Informally, its state space can be partitioned into an initial component and a deterministic accepting component. Once a run enters the deterministic component, all subsequent transitions are uniquely determined by the observed input symbols.

Definition 3.3 (LDBA)**.** An LDBA is denoted by $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, F)$, where Q is the automaton state space, $\Sigma = 2^{\mathcal{AP}}$ is the input alphabet, δ is the transition relation, q_0 is the initial state, and F denotes the Büchi acceptance set. Its state space can be partitioned as

$$Q = Q_N \cup Q_D,$$

where Q_N contains the potentially nondeterministic states and Q_D contains the deterministic component. The deterministic component satisfies the following relation and transitions from Q_D remain within Q_D.

$$|\delta(q, \sigma)| \leq 1, \qquad \forall q \in Q_D,\ \sigma \in \Sigma$$

In many LDBA constructions, the transition from the initial component Q_N to the deterministic component Q_D is represented by an **ϵ-transition**,

$$q \xrightarrow{\epsilon} q', \tag{3.6}$$

which changes the automaton state without consuming a symbol from the input trace. Thus, unlike an ordinary transition $q \xrightarrow{\sigma} q'$, an ϵ-transition does not correspond to a change in the observed environment state or propositional valuation.

Interpretation in reinforcement learning. The presence of ϵ-transitions introduces an additional issue when the automaton is composed with an MDP. A standard MDP transition is induced by an environment action and therefore changes the physical state according to the transition kernel. An ϵ-transition, however, changes only the automaton state.

Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mu_0, L)$ be a labelled MDP and let $\mathcal{A}_\varphi$ be an LDBA. The product state space remains $\mathcal{S}_\otimes = \mathcal{S} \times Q$. To represent the nondeterministic automaton choices, the action space of the product MDP can be augmented with additional actions corresponding to available ϵ-transitions:

$$\mathcal{A}_\otimes = \mathcal{A} \cup \mathcal{A}_\epsilon,$$

where $\mathcal{A}_\epsilon = \{\epsilon_{q'} \mid q \xrightarrow{\epsilon} q' \text{ is available from the current automaton state}\}$.

For an ordinary environment action $a \in \mathcal{A}$, the product transition is given by

$$P_\otimes\left((s', q') \mid (s, q), a\right) = \begin{cases} P(s' \mid s, a), & q' \in \delta(q, L(s')), \\ 0, & \text{otherwise.} \end{cases}$$

By contrast, an ϵ-action changes only the automaton state:

$$P_{\otimes}\left((s',q') \mid (s,q),\epsilon_{q'}\right) = \begin{cases} 1, & s' = s \text{ and } q \xrightarrow{\epsilon} q', \\ 0, & \text{otherwise.} \end{cases}$$

Thus, selecting an ϵ-action leaves the physical environment unchanged while resolving a nondeterministic transition in the temporal monitor. This construction transforms the nondeterminism of the LDBA into an explicit decision within the product MDP [37, 18]. Importantly, the ϵ-actions are not actions available in the original environment. They are artificial actions introduced solely to resolve the automaton's nondeterministic choice. The reinforcement-learning agent may therefore be required to learn not only how to act in the physical environment, but also when to commit to a particular automaton branch.

Büchi acceptance. Because an LDBA operates over infinite traces, acceptance is not determined simply by reaching a terminal accepting state. Instead, a run $q_0 q_1 q_2 \cdots$ satisfies the Büchi acceptance condition if states in the acceptance set F are visited infinitely often.

$$\mathrm{Inf}(q_0 q_1 q_2 \cdots) \cap F \neq \emptyset,$$

where $\mathrm{Inf}(\cdot)$ denotes the set of states occurring infinitely often along the run.

Consequently, reinforcement-learning methods based on LDBAs typically construct rewards or acceptance-tracking mechanisms that encourage the policy to reach and repeatedly satisfy the relevant accepting conditions [18, 29].

Implications for task progression. LDBAs provide an attractive compromise between the expressiveness of general LTL and the cost of full determinisation. However, their restricted nondeterminism changes the nature of the interface between temporal logic and reinforcement learning. In a deterministic automaton, observing σ_t determines a unique successor $q_{t+1} = \delta(q_t, \sigma_t)$. For an LDBA, progression may instead require choosing between several automaton transitions, including ϵ-transitions. The temporal progress state is therefore not always determined solely by the current automaton state and observed propositional valuation. This distinction will become important later in this thesis. The LTL_f^+ obligation fragment considered here admits a Deterministic Weak Automaton construction, so its temporal progression can be represented as a deterministic update. This removes the need to encode automaton

nondeterminism as an additional control choice from the agent's learning and provides a cleaner separation between *task progression*, handled by the automaton, and *environment control*, handled by the reinforcement learning policy.

3.2 Linear Temporal Logic on Finite Traces

Classical Linear Temporal Logic (LTL) is defined over infinite execution sequences (ω-words), making it the standard paradigm for verifying non-terminating reactive systems (e.g., operating systems or power grids). However, Reinforcement Learning environments are episodic, that is, they are terminating upon reaching a goal, incurring a fatal failure, or exceeding a finite horizon T. To align formal specifications with episodic learning, we adopt Linear Temporal Logic over Finite Traces (LTL_f) [12]. LTL_f shares the intuitive syntax of LTL but interprets formulas over finite sequences of truth assignments, providing a mathematically sound semantics for episodic task completion.

Definition 3.4. Let $AP = \mathcal{X} \cup \mathcal{Y}$ be the atoms, partitioned into environment (uncontrollable) variables and agent (controllable) variables. An infinite trace is an infinite sequence $\pi \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$. The empty sequence is denoted ϵ.

To bridge the continuous or high-dimensional physical state space $\mathcal{S}$ of an MDP with abstract logical reasoning, we define a set of Atomic Propositions AP. Each proposition $p \in AP$ represents a high-level boolean feature of interest (e.g., at_charger, carrying_object, hazard_detected).

Definition 3.5 (Labeling Function)**.** A **Labeling Function** (or evaluation function) is a mapping

$$L : \mathcal{S} \longrightarrow 2^{AP}$$

which assigns to every environment state $s \in \mathcal{S}$ the subset of atomic propositions $L(s) \subseteq AP$ that hold true in state s.

Through L, an agent's execution trajectory $\tau = s_0, s_1, s_2, \ldots, s_T$ projects onto a finite trace of propositional evaluations $\sigma = L(\tau) = \sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_T$, where each $\sigma_t = L(s_t) \in 2^{AP}$.

LTL_f formulas are constructed from atomic propositions, standard Boolean connectives, and temporal modal operators.

Definition 3.6 (LTL_f Syntax)**.** Given a set of atomic propositions AP, the set of valid LTL_f formulas is generated inductively by the following grammar.

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2 \mid \mathbf{WX}\varphi$$

where $p \in AP$, $\mathbf{X}$ is the **Next** operator, $\mathbf{WX}$ is the **Weak Next** operator, and $\mathbf{U}$ is the **Until** operator.

From these core primitives, we derive standard logical connectives ($\vee$, $\implies$, $\iff$) along with the following key syntactic temporal abbreviations.

- Eventually (Future): $\mathbf{F}\varphi \equiv \top \mathbf{U} \varphi$ (asserts that φ holds at least once in the remaining trace)
- Always (Globally): $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$ (asserts that φ holds at all future timesteps in the trace)
- Weak Until: $\varphi_1 \mathbf{W} \varphi_2 \equiv (\varphi_1 \mathbf{U} \varphi_2) \vee \mathbf{G}\varphi_1$

The truth value of an LTL_f formula φ is evaluated relative to a finite trace $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_T$ (where $\sigma_t \in 2^{AP}$) at a specific time step $i \in \{0, 1, \ldots, T\}$.

Definition 3.7. Given a trace $\sigma \in 2^{AP}$, the satisfaction relation $(\sigma, i) \models \varphi$ (read as trace σ satisfies φ at position i) is defined inductively for $0 \leq i \leq \mathbf{length}(\sigma) = last$:

$$\begin{aligned}
(\sigma, i) &\models p &&\iff p \in \sigma_i \quad \text{for } p \in AP \\
(\sigma, i) &\models \neg\varphi &&\iff (\sigma, i) \not\models \varphi \\
(\sigma, i) &\models \varphi_1 \wedge \varphi_2 &&\iff (\sigma, i) \models \varphi_1 \text{ and } (\sigma, i) \models \varphi_2 \\
(\sigma, i) &\models \mathbf{X}\varphi &&\iff i < last \text{ and } (\sigma, i+1) \models \varphi \\
(\sigma, i) &\models \varphi_1 \mathbf{U} \varphi_2 &&\iff \exists j \in [i, last] \text{ s.t. } (\sigma, j) \models \varphi_2 \text{ and } \forall k \in [i, j-1], (\sigma, k) \models \varphi_1
\end{aligned}$$

A finite trace σ satisfies φ, denoted $\sigma \models \varphi$, if and only if $(\sigma, 0) \models \varphi$. The finite-trace interpretation changes the meaning of these operators at termination. For example, $\mathbf{F}\,\mathsf{goal}$ is satisfied precisely when goal occurs at some position before the trace terminates.

From LTL_f to Finite Automata

An important property of LTL_f is that its formulas define regular languages over finite words. Consequently, every LTL_f formula can be represented by a finite automaton [12]. For a formula φ, define

$$\mathcal{L}(\varphi) = \{\sigma \in (2^{\mathcal{AP}})^* \mid \sigma \models \varphi\}.$$

There exists a deterministic finite automaton $\mathcal{A}_\varphi = (Q, 2^{\mathcal{AP}}, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. The automaton state reached after reading a trace prefix therefore summarizes the information from that prefix that is relevant to determining whether a continuation can satisfy φ. This finite-memory interpretation will later provide the basis for representing temporal task progress within an MDP.

3.3 Extending LTL_f to Infinite Traces

Although LTL_f provides an effective specification language for finite executions, not every temporal requirement of interest terminates with a finite episode. Safety requirements may need to hold indefinitely, and an autonomous system may repeatedly receive and complete finite-horizon objectives while continuing to operate. Classical LTL captures such infinite-horizon behaviour, but its automata-theoretic treatment generally requires automata over infinite words with correspondingly richer acceptance conditions. LTL_f^+ was introduced by Aminof et al [3] to bridge these perspectives. Rather than interpreting LTL_f itself over infinite traces, LTL_f^+ uses finite-trace LTL_f properties as building blocks for defining properties of infinite executions.

Let $\pi = \pi_0\pi_1\pi_2\cdots \in (2^{\mathcal{AP}})^\omega$ be an infinite trace. For $i \geq 0$, denote its finite prefix through position i by $\pi_{\leq i} = \pi_0\pi_1\cdots\pi_i$. Given an LTL_f formula Φ, LTL_f^+ allows us to reason about how often prefixes of π satisfy Φ.

Definition 3.8 (LTL_f^+ [3])**.** The syntax of LTL_f^+ is given by the following grammar.

$$\Psi := \forall\Phi \mid \exists\Phi \mid \forall\exists\Phi \mid \exists\forall\Phi \mid \Psi \vee \Psi \mid \Psi \wedge \Psi \mid \neg\Psi$$

where Φ are formulas in LTL_f over AP. Common abbreviations are used, such as $\Psi \supset \Psi'$ instead of $\neg\Psi \vee \Psi'$. The four prefix quantifiers correspond to different patterns of satisfaction of the finite-trace property Φ along an infinite execution.

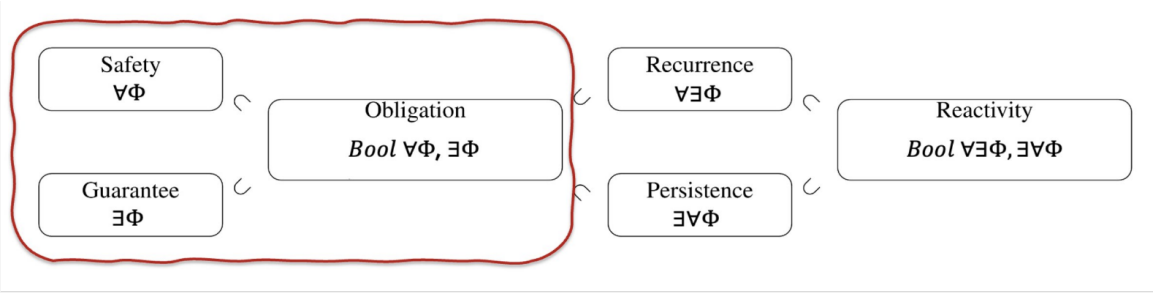


Figure 3.2: The Manna-Pnueli Hierarchy. [Taken from lecture material used by Dr. Giuseppe de Giacomo at University of Oxford]

$$\pi \models \forall\Phi \iff \forall i \geq 0,\ \pi_{\leq i} \models \Phi, \tag{3.7}$$

$$\pi \models \exists\Phi \iff \exists i \geq 0,\ \pi_{\leq i} \models \Phi, \tag{3.8}$$

$$\pi \models \forall\exists\Phi \iff \forall i \geq 0,\ \exists j \geq i : \pi_{\leq j} \models \Phi, \tag{3.9}$$

$$\pi \models \exists\forall\Phi \iff \exists i \geq 0,\ \forall j \geq i : \pi_{\leq j} \models \Phi. \tag{3.10}$$

Intuitively, these operators state that Φ holds on every prefix, some prefix, infinitely many prefixes, or all but finitely many prefixes, respectively. Following the safety-progress hierarchy of Manna and Pnueli, as shown in Figure 3.2, these constructions correspond respectively to safety, guarantee, recurrence, and persistence properties. From the LTL_f^+ formulas, $[\forall\Phi]$ is called a *safety* property, $[\exists\Phi]$ is called a *guarantee* property, $[\exists\forall\Phi]$ is called a *persistence* property, $[\forall\exists\Phi]$ is called a *recurrence* property, and $[\Psi]$ is a *reactivity* property. Arbitrary Boolean combinations yield the general reactivity level.

A key theoretical property of LTL_f^+ is that it does not lose the expressive power of classical LTL. Thus, LTL_f^+ should not be viewed as a finite-trace alternative to LTL. Rather, it provides an alternative decomposition of infinite-trace properties in terms of finite-trace properties, allowing automata constructions for LTL_f to be reused compositionally [3].

3.4 LTL_f^+ Obligations

Although full LTL_f^+ can express arbitrary LTL properties, the present thesis focuses on its *obligation fragment*. Obligation properties occupy the second level of the Manna-Pnueli temporal hierarchy and consist of positive Boolean combinations of safety and guarantee properties [27, 16].

Definition 3.9 (LTL_f^+ Obligations)**.** Formulas in *LTL_f^+ Obligations* is a fragment of LTL_f^+, which are a positive boolean combination of safety and guarantee properties. Thus, the syntax of LTL_f^+ Obligations fragment is given by the following grammar.

$$\Psi := \forall\Phi \mid \exists\Phi \mid \Psi \vee \Psi \mid \Psi \wedge \Psi$$

where Φ are formulas in LTL_f over AP. The semantics remain the same as in LTL_f^+ overall. Note that the negation operator has been dropped since $\forall, \exists$ are dual to each other.

Here, $\forall\Phi$ denotes a safety property: every finite prefix of the infinite execution must satisfy Φ. Conversely, $\exists\Phi$ denotes a guarantee (or co-safety) property: at least one finite prefix must satisfy Φ. The fragment is closed under positive Boolean combinations. Thus, for example, $(\forall\Phi_s) \wedge (\exists\Phi_g)$ requires both an invariant safety requirement and eventual finite evidence of a goal. More generally, $\Psi = (\forall\Phi_1 \vee \exists\Phi_2) \wedge \exists\Phi_3$ combines multiple independently specified finite-trace requirements into a single infinite-trace objective.

Example 3.4.1 (Delivery with Safety and Completion Obligations)**.** *Consider an autonomous delivery agent with atomic propositions $\mathcal{AP} = \{\mathit{key}, \mathit{dock}, \mathit{hazard}\}$. Suppose the agent is required to avoid the hazard region indefinitely while eventually completing a finite delivery sequence in which the key is obtained before the dock is reached. Define the finite-trace formulas*

$$\Phi_{\text{safe}} = \mathbf{G}\neg\mathit{hazard},$$

$$\Phi_{\text{deliver}} = \mathbf{F}\left(\mathit{key} \wedge \mathbf{F}\,\mathit{dock}\right).$$

The corresponding LTL_f^+ obligation specification is $\Psi_{\text{delivery}} = \forall\Phi_{\text{safe}} \wedge \exists\Phi_{\text{deliver}}$. The safety component $\forall\Phi_{\text{safe}}$ requires every finite prefix of the infinite execution to satisfy the finite-trace safety formula. The guarantee component $\exists\Phi_{\text{deliver}}$ requires the existence of some finite prefix witnessing completion of the delivery sequence. Consequently, the complete specification combines a persistent safety requirement with a finitely witnessable task-completion requirement.

Obligation properties are particularly relevant to the setting considered in this thesis because they capture combinations of two common classes of sequential requirements: conditions that must never be violated and conditions for which successful completion can eventually be witnessed. At the same time, restricting attention

to this fragment provides stronger automata-theoretic structure than is available for arbitrary LTL_f^+ formulas.

Automata for LTL_f^+ Obligations

Let $\Psi = B(\forall\Phi_1, \ldots, \forall\Phi_m, \exists\Phi_{m+1}, \ldots, \exists\Phi_n)$ be an LTL_f^+ obligation formula, where B is a positive Boolean combination and each Φ_i is an LTL_f formula. The construction begins by translating each finite-trace component Φ_i into a DFA $\mathcal{A}_i = (Q_i, 2^{\mathcal{AP}}, \delta_i, q_i^0, F_i)$. These component automata track whether the corresponding finite-trace properties hold on prefixes of the observed execution. Their state spaces can then be combined compositionally to construct a deterministic weak automaton representing the overall obligation property [16].

Definition 3.10 (DFA)**.** A **Deterministic Finite Automaton (DFA)** recognizing $\mathcal{L}(\varphi)$ is a 5-tuple $\mathcal{A}_\varphi = \langle Q, \Sigma, \delta, q_0, F\rangle$ where

- Q is a finite set of automaton states,
- $\Sigma = 2^{\mathcal{AP}}$ is the input alphabet,
- $\delta : Q \times \Sigma \to Q$ is the deterministic transition function,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states.

While general DFAs can represent any regular language, real-world task obligations (e.g., safety commitments, sequential deadlines, reachability obligations) exhibit a weak structural recurrence property: task progress is strictly monotonic toward either terminal obligation fulfillment or non-recoverable obligation failure. To exploit this structure for efficient online monitoring and reward shaping, we combine the compiled DFA for each component into a Deterministic Weak Automaton (DWA).

Definition 3.11 (DWA)**.** A **Deterministic Weak Automaton (DWA)** is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, where:

- Σ is a finite alphabet;
- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $\delta : Q \times \Sigma \to Q$ is a deterministic transition function;

- $F \subseteq Q$ is a set of accepting states.

The automaton is **weak** if there exists a partition of Q into disjoint sets $Q_1, \ldots, Q_n$ such that:

1. For each Q_i, either $Q_i \subseteq F$ or $Q_i \cap F = \emptyset$. Components where $Q_i \subseteq F$ are called **accepting components**, and others are **rejecting components**.

2. There is a partial order $\leq$ on the components such that for every $q \in Q_i$ and $a \in \Sigma$, if $\delta(q, a) = q'$, then $q' \in Q_j$ with $Q_j \leq Q_i$.

A run $\rho = q_0, q_1, \ldots$ is **accepting** if the set of states visited infinitely often, $\mathrm{Inf}(\rho)$, is a subset of F. Due to the weak structure, this is equivalent to saying that the run eventually settles in an accepting component $Q_i \subseteq F$.

The weakness condition restricts transitions between strongly connected components to a partial-order structure. Consequently, once a run leaves a component, it cannot later return to it through a cycle involving a different component. This structure is considerably simpler than the acceptance mechanisms required for arbitrary ω-regular properties. For LTL_f^+ obligations, the resulting DWA provides a deterministic finite-state monitor of progress through the infinite-trace objective. This deterministic structure will be particularly important in later chapters, where the automaton state is used as the basis for a representation of temporal task progress.

3.5 MDPs for Temporal Goals

The preceding sections introduced temporal formulas as specifications over traces and automata as finite-state representations of their temporal structure. We now connect these logical representations to the sequential decision-making framework introduced in the previous chapter.

We model the underlying physical environment as a Labelled Markov Decision Process without a reward function as the following tuple.

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, L \rangle$$

where:

- $\mathcal{S}$ is the set of physical environment states.

- $\mathcal{A}$ is the set of agent actions.

- $T : \mathcal{S} \times \mathcal{A} \longrightarrow \text{Prob}(\mathcal{S})$ is the transition function, where $\Pr(s') = P(s' \mid s, a)$ denotes the probability of transitioning to s' upon taking action a in state s.
- $L : \mathcal{S} \longrightarrow 2^{AP}$ is a labeling function mapping each physical state $s \in \mathcal{S}$ to the set of truth values of atomic propositions satisfied at s.

A trajectory $s_0, a_0, s_1, a_1, \ldots$ therefore induces the propositional trace $L(s_0)L(s_1)\cdots$. Note that no reward function is included as we want our reward to come from a LTL_f^+ goal φ. Our objective is to find a strategy $\sigma : \mathcal{S}^* \longrightarrow \mathcal{A}$ that maximizes the probability of satisfying φ. For a temporal specification Ψ, the objective is no longer naturally described by an immediate scalar reward. Instead, the quality of a policy can be defined in terms of the probability that the induced trace satisfies the temporal property.

$$J_\Psi(\pi) = \Pr_\pi [L(\tau) \models \Psi] \tag{3.11}$$

The corresponding control objective is $\pi^* \in \arg\max_\pi J_\Psi(\pi)$. The principal difficulty is that satisfaction of Ψ may depend on the history of the trajectory. Consequently, two trajectories that reach the same physical state s_t may differ in which temporal requirements remain active. The environment state alone therefore need not provide a Markovian representation of the temporal decision problem.

3.5.1 Automata as Finite Task Memory

Let $\mathcal{A}_\Psi = (Q, 2^{\mathcal{AP}}, \delta, q_0, F)$ be a deterministic automaton monitoring the temporal specification. Given the sequence of environment labels, its state evolves according to $q_{t+1} = \delta(q_t, L(s_{t+1}))$. Thus, q_t is a deterministic function of the observed propositional history. Rather than retaining the entire trajectory $L(s_0), L(s_1), \ldots, L(s_t)$, the agent can retain the finite automaton state q_t, which summarizes the history relevant to the temporal objective. The augmented state is as follows.

$$\tilde{s}_t = (s_t, q_t) \in \mathcal{S} \times Q$$

Definition 3.12 (Product MDP)**.** Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, L \rangle$ be a Labelled Markov Decision Process, where $\mathcal{S}$ is the physical state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \longrightarrow \Pr(\mathcal{S})$ is the physical transition probability kernel, and $L : \mathcal{S} \longrightarrow 2^{\mathcal{AP}}$ is a labeling function mapping states to sets of satisfied atomic propositions.Let $\mathcal{A}_\varphi = \langle Q, 2^{\mathcal{AP}}, \delta, q_0, F \rangle$ be a Deterministic Automaton monitoring specification φ, where Q is a finite set of automaton states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times 2^{AP} \longrightarrow Q$ is the deterministic transition

function. The **Product Markov Decision Process (Product MDP)** is defined as the following tuple.

$$\mathcal{M}_{\mathcal{P}} = \mathcal{M} \times \mathcal{A}_{\varphi} = \langle \mathcal{S}_{\mathcal{P}}, \mathcal{A}, T_{\mathcal{P}}, R_{\mathcal{P}}, s_{\mathcal{P},0}, \gamma, F_{\mathcal{P}} \rangle$$

where:

- $\mathcal{S}_{\mathcal{P}} = \mathcal{S} \times Q$ is the **product state space**, where each product state $s_{\mathcal{P}} = (s, q)$ pairs a physical state $s \in \mathcal{S}$ with an automaton task state $q \in Q$.
- $\mathcal{A}$ is the shared **action space**.
- $T_{\mathcal{P}} : \mathcal{S}_{\mathcal{P}} \times \mathcal{A} \longrightarrow \mathcal{P}(\mathcal{S}_{\mathcal{P}})$ is the **synchronized product transition kernel**, defined such that the probability of transitioning to state (s', q') given action a in state (s, q) is:

  $$T_{\mathcal{P}}((s', q') \mid (s, q), a) = \begin{cases} T(s' \mid s, a) & \text{if } q' = \delta(q, L(s')) \\ 0 & \text{otherwise} \end{cases}$$

  where $T(s' \mid s, a) \equiv [T(s, a)](s')$ represents the physical transition probability to s', and q' updates deterministically via δ evaluated on the destination label $L(s')$.
- $R_{\mathcal{P}} : \mathcal{S}_{\mathcal{P}} \times \mathcal{A} \times \mathcal{S}_{\mathcal{P}} \longrightarrow \mathbb{R}$ is the **product reward function**, constructed from task satisfaction signals and automaton state transitions.
- $s_{\mathcal{P},0} = (s_0, \delta(q_0, L(s_0))) \in \mathcal{S}_{\mathcal{P}}$ is the **initial product state**, initialized given initial physical state $s_0 \in \mathcal{S}$.
- $\gamma \in (0, 1)$ is the **discount factor**.
- $F_{\mathcal{P}} = \mathcal{S} \times F \subseteq \mathcal{S}_{\mathcal{P}}$ is the set of **accepting product goal states**.

The product construction restores the information required to distinguish physical states reached under different temporal histories. In particular, the transition distribution of the augmented state depends only on the current product state and action.

$$\Pr\left(S_{t+1} = s', Q_{t+1} = q' \mid H_t, S_t = s, Q_t = q, A_t = a\right) = P_{\otimes}\left((s', q') \mid (s, q), a\right) \quad (3.12)$$

Hence, provided that the automaton state captures the history relevant to the temporal objective, the augmented process is Markovian. This observation provides the central connection between temporal logic and reinforcement learning; the automaton state acts as a finite progress representation through which a history-dependent task can be incorporated into a Markovian learning problem.

Chapter 4

Multi-Task Reinforcement Learning with LTL_f^+ Obligations

The preceding chapters established two complementary foundations for the setting considered in this thesis. Chapter 2 introduced Multi-Task Reinforcement Learning (MTRL), in which a single policy is conditioned on a task representation and trained to operate across multiple objectives. Chapter 3 subsequently showed how temporally extended objectives can be represented using temporal logic and how automata can provide finite memory of task progress. Combining these two perspectives introduces a representational challenge. For a single temporal specification, it is sufficient to augment the environment state with any task memory that correctly records the history relevant to satisfaction. In the multi-task setting, however, the progress representation must serve an additional purpose: it must provide a *shared interface* through which a single policy can reason about many different temporal specifications. Let $\mathcal{T}$ denote a temporal task and let $\rho_t^{\mathcal{T}}$ denote a representation of the progress made towards that task at timestep t. A task-conditioned policy may then be written as $\pi_\theta\left(a_t \mid s_t, \rho_t^{\mathcal{T}}\right)$. For an individual task, $\rho_t^{\mathcal{T}}$ need only contain sufficient information to make the augmented decision process Markovian. For MTRL, however, representations produced for different tasks must additionally expose enough common structure for experience to transfer between them. Two task states requiring similar future behaviour should ideally admit similar representations even when they originate from independently constructed specifications.

This chapter examines existing approaches to representing temporally extended tasks in reinforcement learning from this perspective. We first review methods that expose non-Markovian reward and task structure to the learning agent, including reward machines and temporal-logic-based task representations. We then consider approaches designed explicitly for generalisation across temporal specifications, ranging

from modular decomposition to formula- and automata-conditioned policies. These approaches motivate the notion of a *Universal Progress Interface* (UPI): an interface responsible for maintaining task progress and exposing it to a shared reinforcement learning policy. We examine two principal mechanisms for constructing such an interface and identify the representational and computational trade-offs associated with each. However, the use of temporal logic in reinforcement learning predates recent formula-conditioned multi-task approaches. Earlier methods typically translated temporal specifications into automata and composed these automata with the environment, allowing temporal satisfaction to be represented through an augmented Markov state [34, 18, 7].

The primary conclusion of the chapter is that neither the syntax of a progressed formula nor an opaque automaton-state identifier simultaneously provides all of the properties desirable in the multi-task setting. The former exposes semantic structure but may yield representation-level redundancy, while the latter provides compact and efficient memory but typically obscures the meaning of the remaining task. This observation motivates the semantic progress representations developed in the following chapter.

4.1 Temporal Task Representations in Reinforcement Learning

Standard reinforcement learning commonly represents an objective through a scalar reward function. For many temporally extended tasks, however, the reward associated with an event depends on what has occurred previously. Such objectives can be viewed as *non-Markovian rewards*, that is, the reward or task status cannot be determined solely from the current physical state and action. A general approach to this problem is to augment the physical state with an explicit representation of task history. Rather than requiring the policy to reconstruct the relevant history from observations, a separate symbolic or finite-state mechanism maintains the portion of the execution relevant to the objective. This perspective has appeared in several forms across the reinforcement learning literature. Temporal-logic automata provide one such mechanism, but related approaches based on finite-state reward representations and task decomposition share the same underlying principle of exposing the structure of the objective to the learning algorithm rather than treating the reward function as an opaque scalar signal.

4.1.1 Reward Machines and Structured Reward Representations

Reward Machines (RMs) provide a finite-state representation of structured, possibly non-Markovian reward functions [20, 21, 45]. A reward machine maintains an internal state that changes in response to high-level events observed in the environment. Each machine transition can additionally emit a reward or select an associated reward function. A reward machine may be represented abstractly as $\mathcal{R} = (U, u_0, \delta_u, \delta_r)$, where U is a finite set of reward-machine states, u_0 is the initial state, δ_u determines how the machine state changes in response to high-level events, and δ_r specifies the reward associated with a machine transition.

Given an event labelling function $L : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to 2^{\mathcal{AP}}$, the reward-machine state evolves according to

$$u_{t+1} = \delta_u \left(u_t, L(s_t, a_t, s_{t+1})\right) \tag{4.1}$$

The resulting augmented process has state $\tilde{s}_t = (s_t, u_t)$, closely paralleling the automata-product construction introduced in the previous chapter.

The principal contribution of reward machines is not merely the restoration of Markovianity. By exposing reward structure explicitly, they enable learning algorithms to exploit task decomposition, reward shaping, and counterfactual experience. Icarte et al. show that this additional structure can substantially improve sample efficiency compared with treating the reward function as a black box [21].

A closely related approach is provided by the *restraining bolt* framework of De Giacomo et al. [11]. Rather than specifying a structured reward function directly as a finite-state machine, restraining bolts allow desired agent behaviour to be expressed declaratively using LTL_f or LDL_f specifications. The temporal specification is compiled into a finite-state automaton whose state records the stage reached in satisfying the specification. This automaton state is then exposed to the reinforcement-learning agent as additional state information, allowing the agent to distinguish histories that may be identical from the perspective of the underlying MDP state but differ with respect to progress towards the temporal objective. An important feature of the restraining-bolt formulation is the separation between the agent's representation of the environment and the symbolic representation used to specify the desired behaviour. De Giacomo et al. consider an RL agent operating over low-level environment features while the restraining specification is defined over a separate set of high-level symbolic features. The automaton induced by the temporal specification, together

with additional rewards associated with its progression, allows the learned behaviour to conform to the high-level specification without requiring the two representations to be explicitly identified [11].

Reward machines, restraining bolts, and temporal-logic automata therefore share an important conceptual role: each introduces explicit finite memory through which temporally structured objectives can be incorporated into an otherwise Markovian learning problem. Their emphasis, however, differs. Reward machines begin from a structured reward representation, while restraining bolts begin from declarative finite-trace temporal specifications and compile these specifications into automata used during learning. The present thesis follows the latter temporal-logic perspective, but additionally considers how the *semantic content* of an automaton state can be represented in a form suitable for transfer across multiple temporal tasks.

4.1.2 Temporal Logic as a Task Interface

Temporal logic provides an alternative interface in which tasks are specified declaratively rather than directly as reward machines. A formula describes the desired set of executions independently of the dynamics of a particular environment, while a labelling function connects physical states to the propositions referenced by the specification. This separation is attractive in multi-task settings because the same logical task can, in principle, be interpreted across different environments sharing the relevant propositions. Temporal specifications also admit explicit composition through Boolean and temporal operators, allowing large task families to be generated systematically from common sub-goals.

A substantial literature has consequently investigated the integration of temporal specifications with reinforcement learning. Early methods often construct a product between an MDP and an automaton derived from the specification, while later approaches increasingly exploit the internal structure of the specification itself to improve learning or transfer [18, 49, 42]. For example, Yuan et al. represent temporal structure using a finite-state machine and construct an online synchronized product with the underlying MDP, using a modular actor–critic architecture to learn low-level control policies [49]. Such approaches demonstrate that temporal task structure can be exploited during learning rather than serving only as an external verification mechanism.

4.2 Universal Progress Interface

In multi-task RL with temporal specifications, the environment state alone is insufficient to determine the status of the task because satisfaction depends on the sequence of propositions observed so far. A Universal Progress Interface (UPI) provides the additional task memory required to recover Markovian decision making over the augmented state space. Rather than requiring the policy itself to infer temporal progress from an unbounded interaction history, the UPI maintains an explicit task representation ρ_t and updates it whenever a new propositional observation is received.

Let $\sigma_t \subseteq \mathcal{AP}$ denote the set of atomic propositions true at timestep t. Abstractly, the UPI can be viewed as an update-and-encode mechanism

$$(\rho_{t+1}, z_{t+1}) = \mathcal{U}(\rho_t, \sigma_{t+1}), \tag{4.2}$$

where ρ_t is the symbolic or discrete representation of task progress and $z_t \in \mathbb{R}^d$ is the policy-facing numerical representation derived from it. A universal policy is then conditioned jointly on the physical state and task representation,

$$a_t \sim \pi_\theta(\cdot \mid s_t, z_t). \tag{4.3}$$

This decomposition makes explicit that progress tracking and action selection solve different problems. The UPI summarizes the history relevant to the specification, whereas the policy determines how to act given the current physical state and the remaining task.

Three requirements are particularly important in the multi-task setting. First, the representation must provide a common interface across specifications so that a single policy can be trained over many tasks. Second, it must update task progress faithfully as new observations arrive. Third, the online update should remain sufficiently inexpensive for use at every environment step. The following subsections examine three realizations of ρ_t: a residual formula, an automaton state, and an automaton state augmented with semantic information. Their strengths and limitations motivate the obligation-based semantic representation used later in this chapter.

4.2.1 Formula Progression Approach

The formula progression [5, 4] paradigm realizes the UPI by updating task specifications directly at the logical syntax level as environment observations unfold. Under this framework, given an LTL formula φ_t and an observed propositional evaluation

$\sigma_t \subseteq \mathcal{AP}$ at step t, the interface applies expansion laws, such as the fundamental identity for the until operator $\varphi_1 \mathcal{U} \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \mathcal{X}(\varphi_1 \mathcal{U} \varphi_2))$, to evaluate the immediate truths and rewrite the task into a simplified residual formula $\varphi_{t+1} = \text{prog}(\varphi_t, \sigma_t)$. This updated formula encapsulates the exact logical requirements that must be satisfied over the remainder of the trace. The policy is subsequently conditioned on a neural embedding of the progressed syntax tree, allowing the network to adapt its behavior as sub-goals are accomplished and collapsed into Boolean constants ($\top$ / $\bot$).

The primary advantage of formula progression lies in its simplicity and directness. Because progression operates natively on syntax, it eliminates the need to compile global transition structures or automata offline prior to policy execution. For short-horizon tasks or restricted syntactic fragments (such as co-safety formulas), this dynamic, on-the-fly rewriting offers a lightweight mechanism for tracking task completion. However, formula progression suffers from critical theoretical and computational limitations when applied to general multi-task RL. Most notably, plain formula progression exhibits an expressiveness bottleneck: it cannot track acceptance conditions for infinite-horizon properties, such as recurrence ($\Box \diamond \psi$) or persistence ($\diamond \Box \psi$). Because progression evaluates formulas purely step-by-step without maintaining explicit global structural memory, it cannot distinguish between infinite executions that satisfy infinite-horizon conditions versus those that stall. Furthermore, repeatedly applying expansion rules leads to syntactic blowup and redundancy; formulas grow increasingly complex over time, producing distinct syntactic trees that share identical semantic meanings, which degrades the quality of downstream neural embeddings and hinders generalizability.

LTL2Action

An alternative to explicitly constructing a task automaton is to condition the policy directly on a logical representation of the remaining task. Formula progression provides a natural mechanism for maintaining such a representation. Given a temporal formula φ_t and an observed propositional valuation σ_t, progression computes a residual formula $\varphi_{t+1} = \text{prog}(\sigma_t, \varphi_t)$ such that satisfaction of the original formula after observing σ_t is equivalent to satisfaction of the residual formula over the remaining trace. For example, the expansion rule

$$\varphi_1 \mathbf{U} \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \mathbf{X}(\varphi_1 \mathbf{U} \varphi_2))$$

allows the current observation to determine which portion of an until obligation remains active.

LTL2Action uses this idea to construct a task-conditioned RL formulation in which the progressed LTL instruction becomes part of the augmented state [42]. A neural LTL module encodes the syntax tree of the current residual specification and conditions the reinforcement learning policy on the resulting representation. This design has an important advantage for multi-task generalisation: different tasks are represented through a common compositional language rather than unrelated categorical identifiers. LTL2Action demonstrates zero-shot generalisation to instructions not observed during training and introduces an environment-independent pretraining procedure for learning representations of LTL syntax [42]. The approach nevertheless illustrates several limitations of purely syntax-based task representations. First, the learning formulation used by LTL2Action focuses on co-safe specifications for which satisfaction can be established after a finite prefix. General infinite-horizon properties such as recurrence require additional acceptance information beyond simply detecting progression to $\top$ or $\bot$. Second, logically equivalent formulas need not possess identical syntax trees. Repeated progression and simplification can therefore expose the neural encoder to multiple syntactic representations of closely related or equivalent residual tasks. The policy must learn to identify such semantic relationships indirectly from syntactic structure. Formula progression consequently provides a semantically meaningful and compositional task representation, but does not by itself provide the compact canonical progress state supplied by an automaton.

4.2.2 Automata-Theoretic Approach

To overcome the expressiveness limitations of syntactic rewriting, the automata-theoretic paradigm compiles the LTL formula φ into an equivalent state-machine representation (typically a variant of a Büchi automaton, such as a Limit-Deterministic Büchi Automaton (LDBA)) before or during execution. Under this approach, the specification is instantiated as an automaton $\mathcal{A}_\varphi = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q denotes the finite set of automaton states, $\Sigma = 2^{\mathcal{AP}}$ is the propositional alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, q_0 is the initial state, and F represents the set of accepting states. The UPI tracks execution by advancing the automaton state $q_{t+1} = \delta(q_t, \sigma_t)$ based on environment observations, feeding an encoding of q_t to the policy alongside the environment state s_t.

The main strength of the automata-theoretic framework is its theoretical completeness and full expressiveness across all of LTL. By incorporating explicit acceptance sets F and capturing infinite-horizon semantics through automaton loops, automata naturally model complex temporal behaviors, including safety, guarantee, recurrence, and persistence. The discrete automaton state provides a minimal, sufficient memory representation required to resolve the history dependency of the task. Despite these advantages, standard automata-theoretic representations present significant challenges for multi-task reinforcement learning. Traditional state encodings, such as one-hot vectors or arbitrary state indices, lack cross-task semantic meaning, causing the policy to become tightly coupled to specific automaton structures and preventing zero-shot generalization to unseen tasks. Moreover, compiling full LTL formulas into deterministic or limit-deterministic automata can suffer from worst-case exponential state space explosions.

DeepLTL

DeepLTL addresses generalisation over full LTL specifications by deriving structured plans from Büchi automata rather than conditioning the policy on raw automaton states [22]. For a current automaton state, the method extracts *reach–avoid sequences*: ordered sequences describing propositions that should be reached while avoiding specified conditions. Such a sequence constitutes a candidate plan for producing an accepting run. A learned policy is conditioned on an embedding of the selected sequence, $\pi_\theta\left(a_t \mid s_t, e(\varsigma)\right)$, where ς denotes a reach–avoid sequence.

Because multiple sequences may be feasible from the same automaton state, DeepLTL additionally learns a critic that estimates which candidate plan is most appropriate in the current environment. This allows the method to support both finite- and infinite-horizon LTL specifications and to perform zero-shot execution of specifications not encountered during training [22]. The representation is richer than a raw automaton-state identifier because it exposes an explicit future plan. However, this richness introduces a different computational cost. Candidate accepting paths must be obtained from the automaton, and selecting a representation requires reasoning over the possible reach–avoid sequences. The SemLTL authors identify this search as an important scaling bottleneck: the complete LDBA must be available and accepting paths must be explored in order to identify the preferred sequence.

4.2.3 Compositional and Modular Approaches

One family of methods attempts to decompose complex temporal objectives into smaller reusable behaviours. Instead of conditioning directly on a complete specification, the agent learns policies for primitive propositions or subtasks and combines these behaviours when solving more complex instructions. Such decomposition can substantially improve transfer when tasks are formed from a stable vocabulary of reusable components. However, decomposing a temporal specification into independently optimised local objectives may discard information about dependencies between future sub-goals. An action that efficiently achieves the next proposition need not be optimal with respect to the complete specification when the choice of how that sub-goal is reached affects later requirements. This tension between *local progress* and *global task structure* is central to the representation problem considered in this thesis. A useful progress interface should expose actionable sub-goal information without reducing the remaining temporal specification to a sequence of independent myopic objectives.

4.3 Towards Semantic Labelled Automata for Representations

The preceding comparison reveals that task monitoring and task representation need not be solved by the same object. Formula progression uses a single object (the residual formula) for both: the formula records progress and simultaneously communicates the remaining task to the policy. This preserves semantic information, but exposes the policy directly to syntactic variation. A conventional automaton separates execution history from the original formula more aggressively. The automaton state provides an efficient progress monitor, but a raw state identifier communicates almost nothing about the meaning of that progress to a policy trained across tasks. This suggests a third design principle:

> *Use an automaton to track progress, but expose a semantic description of the automaton state to the learning agent.*

Under this decomposition, the internal progress state is $\rho_t = q_t$, while the policy representation is derived through an additional semantic mapping $\ell : Q \to \mathcal{L}$, followed by an encoder $E : \mathcal{L} \to \mathbb{R}^d$. The resulting task representation is therefore $z_t = E(\ell(q_t))$. This architecture retains the automaton as the mechanism responsible for

history tracking while allowing the policy to reason about the logical meaning of the current task state. Recent work on Semantically Labelled Automata demonstrates this principle for multi-task reinforcement learning with full LTL specifications [1].The following chapter examines this construction in detail and develops its counterpart for the LTL_f^+ obligation setting considered in this thesis.

Table 4.1: Comparison of progression paradigms for the Universal Progress Interface (UPI) in multi-task reinforcement learning with temporal specifications.

Dimension	**Formula Progression**	**Automata Thoeretic**
Expressiveness & Scope	**Limited.** Primarily suited to finite-horizon co-safety (LTL_f); cannot naturally capture recurrence ($\Box \diamond \psi$) or persistence ($\diamond\Box\psi$).	**Full LTL.** Captures all ω-regular properties, including safety, liveness, and recurrence, through acceptance conditions (e.g., LDBAs).
Zero-Shot Generalization	**Moderate.** Can generalize across related subformulas, but is sensitive to syntactic variation and progression-tree growth.	**Poor.** Discrete state representations (e.g., indices or one-hot vectors) provide no semantic correspondence across tasks.
Memory Representation	**Syntactic.** Dynamic syntax trees are updated online via $\text{prog}(\varphi, \sigma_t)$.	**Discrete.** Opaque automaton states $q_t \in Q$ encode the relevant execution history.
Offline Overhead	**Low.** No automaton compilation is required; progression rules are applied directly at runtime.	**High.** LTL-to-automaton compilation can incur exponential state-space growth.
Online Overhead	**High / Variable.** Progressed syntax trees may grow over time, increasing per-step representation and evaluation costs.	**Low.** State updates require only the transition $q_{t+1} = \delta(q_t, \sigma_t)$.
Sub-Goal Information	**Implicit.** Progression exposes remaining obligations indirectly through subformula reduction.	**Limited.** Raw automaton-state identifiers provide little interpretable sub-goal information without additional shaping.

The comparison in Table 4.1 isolates the design trade-off that is most important for this thesis. Formula progression retains explicit logical meaning but can

produce unstable syntactic representations, whereas a conventional automaton gives compact and efficient progress tracking at the cost of making the task state semantically opaque. Semantically labelled automata suggest a useful middle ground: retain the automaton as the mechanism for tracking execution, but expose a logical description of the residual task to the policy.

Chapter 5

Semantically Labelled Automata

The preceding chapter identified a central representational tension in multi-task reinforcement learning with temporal specifications. Formula-based representations expose the logical meaning of the remaining task, but require the policy to operate directly on syntactic objects whose form may change through progression and simplification. Automata provide a compact finite-state representation of temporal progress, but conventional automaton states are semantically opaque: the identifier assigned to a state conveys little information about the future behaviour required from that state. Semantically Labelled Automata provide a mechanism for separating these two roles. The automaton remains responsible for tracking temporal progress, while each automaton state is augmented with structured semantic information describing the temporal requirements represented by that state. This information can subsequently be transformed into a fixed-dimensional embedding and supplied to a universal reinforcement learning policy.

The method developed in this thesis, which we refer to as $SemLTL_f^+$, retains this central design principle but changes each component required to support the LTL_f^+ obligation fragment. In particular, SemLTL_f^+ replaces the LDBA monitor with a Deterministic Weak Automaton (DWA), adapts the semantic representation to the labels produced by the obligation construction, reduces the semantic feature vector from 386 to 167 dimensions, and simplifies the reinforcement-learning interface by removing the ϵ-transition actions required by the original LDBA-based policy.

The organisation of this chapter follows these changes directly. For each component, we first describe the corresponding mechanism in SemLTL and then introduce the modification made in SemLTL_f^+. Throughout the chapter, it is useful to distinguish three objects:

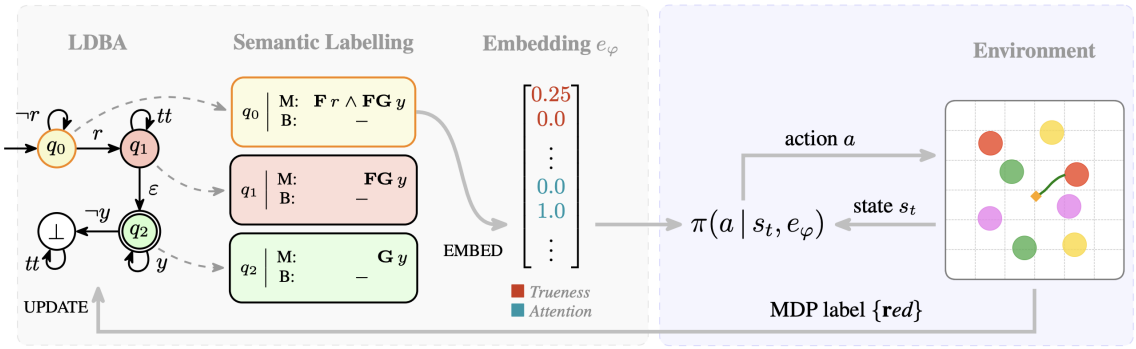


Figure 5.1: SemLTL Pipeline taken from [1].

$$\boxed{\text{automaton state } q_t} \longrightarrow \boxed{\text{semantic label } \ell(q_t)} \longrightarrow \boxed{\text{embedding } z_t}.$$

The automaton state provides finite memory of the execution, the semantic label exposes the logical meaning associated with that state, and the embedding converts this symbolic information into a fixed-dimensional numerical representation consumed by the policy. SemLTL$_f^+$ preserves this separation while changing how each object is constructed.

5.1 Overview of SemLTL and SemLTL$_f^+$

The central insight of SemLTL is that the representation used to *monitor* a temporal task need not be identical to the representation exposed to the learning algorithm [1]. Consider an automaton $\mathcal{A}_\varphi$ constructed for a temporal specification φ. Within a single automaton, a state q is meaningful through its outgoing transitions and acceptance structure. Its numerical identifier, however, is arbitrary. Consequently, a representation such as onehot(q) distinguishes states within one task but provides no intrinsic relation between semantically similar states belonging to different task automata.

SemLTL therefore augments automaton states with logical information generated by a semantic LTL-to-automata translation. Abstractly, a semantically labelled automaton can be represented as

$$\mathcal{A}_\varphi^{\text{sem}} = (\mathcal{A}_\varphi, \ell)\,, \qquad \ell : Q \to \mathcal{L}_{\text{sem}},$$

where $\ell(q)$ denotes the semantic information associated with state q. A numerical representation is computed only afterwards, $z(q) = E(\ell(q)) \in \mathbb{R}^d$. This distinction

Table 5.1: Structural differences between SemLTL and SemLTL$_f^+$.

Component	**SemLTL**	**SemLTL$_f^+$**
Specification language	Full LTL	LTL$_f^+$ obligation fragment
Underlying monitor	Limit-Deterministic Büchi Automaton	Deterministic Weak Automaton
Automaton choice	Restricted nondeterminism with possible ϵ-transitions	Fully deterministic progression
Acceptance structure	Büchi acceptance	Weak ω-acceptance
Semantic state information	Main and, when required, breakpoint formulae	Single semantic label associated with the DWA state
Embedding	Features extracted from main and breakpoint formulae and concatenated	Features extracted once from the DWA semantic label
Feature-vector size	386	167
Policy action space	Environment actions and available ϵ-actions	Environment actions only
Progress update	Environment transition or policy-selected ϵ-transition	Unique DWA transition induced by the observed label

is retained in SemLTL$_f^+$. What changes is the source of the automaton and, consequently, the structure of the semantic labels. SemLTL begins with a full LTL formula and constructs a semantically labelled LDBA. SemLTL$_f^+$ instead begins with an LTL$_f^+$ obligation and constructs a DWA from its finite-trace components. The semantic embedding is then computed from the labels associated with the DWA states. Table 5.1 summarises the principal architectural differences.

5.2 Automaton Construction

5.2.1 Semantically Labelled LDBAs

SemLTL supports full LTL and therefore requires an automaton capable of representing arbitrary LTL properties over infinite traces. It uses a Limit-Deterministic Büchi Automaton (LDBA), written as

$$\mathcal{B}_\varphi = (Q, \Sigma, \delta, \mathcal{E}, q_0, F),$$

where Q is a finite set of states, $\Sigma = 2^{AP}$ is the propositional alphabet, $\delta : Q \times \Sigma \to Q$ is the ordinary transition function, $\mathcal{E} : Q \to 2^Q$ contains the available ϵ-transitions,

q_0 is the initial state, and $F \subseteq Q$ is the Büchi accepting set [37, 1].

An LDBA admits a partition

$$Q = Q_I \uplus Q_A,$$

where Q_I is the initial component and Q_A is the deterministic accepting component. Ordinary transitions remain within their current component, while ϵ-transitions may move from Q_I to Q_A. Once the accepting component has been entered, execution remains deterministic. A run is accepting when states in F are visited infinitely often.

The semantic LTL-to-LDBA construction used by SemLTL additionally associates logical information with the generated states. SemLTL distinguishes a *main formula*, which captures the remaining temporal requirement, and a *breakpoint formula*, which is required to track progress within recurring obligations [1]. For example, a recurrence requirement such as $\mathbf{GF}r$ does not disappear after one occurrence of r. The main formula therefore remains recurrent, while the breakpoint component tracks progress towards the next occurrence whose completion generates Büchi acceptance.

This structure is important for SemLTL's policy. The current progress state is not always determined solely by the next environment label: when an ϵ-transition is available, the policy must also decide whether to move from the initial component into a candidate state of the accepting component.

5.2.2 SemLTL$_f^+$: Obligation-Based DWA Construction

SemLTL$_f^+$ changes the automaton layer rather than adapting the LDBA construction superficially. The task language is restricted to the LTL$_f^+$ obligation fragment introduced in Chapter 3. This fragment consists of positive Boolean combinations of safety and guarantee properties whose underlying conditions are expressed using LTL$_f$ formulae. The obligation structure admits a deterministic weak-automaton representation [3, 16].

Let an obligation specification be written abstractly as

$$\Psi = B(\forall\Phi_1, \ldots, \forall\Phi_m, \exists\Phi_{m+1}, \ldots, \exists\Phi_n)\,,$$

where each Φ_i is an LTL$_f$ formula and B is a positive Boolean combination. The construction begins from the finite-trace components. Each Φ_i can be represented by a deterministic finite automaton $\mathcal{D}_i = (Q_i, 2^{AP}, \delta_i, q_i^0, F_i)$ recognising precisely the

finite traces satisfying Φ_i. These finite-prefix monitors are then combined according to the obligation structure to obtain a deterministic weak ω-automaton

$$\mathcal{W}_\Psi = (Q, 2^{AP}, \delta, q_0, \mathcal{F}_W).$$

The important operational difference is determinism. For every current state q_t and observed valuation σ_{t+1}, there is a unique successor $q_{t+1} = \delta(q_t, \sigma_{t+1})$. No policy decision is required to determine the automaton transition. This makes the DWA a direct temporal monitor: the environment generates a propositional observation and the monitor advances uniquely. Weakness further constrains the acceptance structure. Every strongly connected component of a weak automaton has a uniform acceptance status. Consequently, once the long-run behaviour of a run is confined to a bottom strongly connected component, acceptance is determined by the status of that component. This is structurally different from the Büchi mechanism used by SemLTL, in which acceptance is represented through infinitely repeated visits to designated accepting states.

The automaton substitution changes three aspects of the original SemLTL interface. First, progress tracking becomes deterministic. SemLTL may expose multiple ϵ-successors from an LDBA state, whereas SemLTL$_f^+$ has exactly one successor for a fixed state and observed environment label. Second, the semantic labelling no longer needs to support the main/breakpoint decomposition required by the LDBA acceptance mechanism. The DWA state already contains the progress information induced by the obligation construction, and SemLTL$_f^+$ associates a single semantic description with that state. Third, the automaton no longer contributes additional actions to the reinforcement-learning problem. This difference becomes particularly important in Section 5.4, where the SemLTL hybrid action distribution can be replaced by an ordinary policy over the environment action space. The overall change can therefore be summarised as

$$\underbrace{\text{LTL} \to \text{semantically labelled LDBA} \to \text{environment}/\epsilon\text{ progression}}_{\text{SemLTL}}$$

$$\Downarrow$$

$$\underbrace{\text{LTL}_f^+\text{ obligation} \to \text{LTL}_f\text{ component automata} \to \text{DWA} \to \text{deterministic progression}}_{\text{SemLTL}_f^+}.$$

5.3 Semantic Labels and Task Embeddings

Changing the monitor does not remove the central motivation of SemLTL. A DWA state identifier remains an arbitrary categorical symbol and is therefore no more suitable for cross-task generalisation than a raw LDBA state identifier. SemLTL$_f^+$ consequently retains the semantic-embedding principle: the automaton tracks task progress, while the policy receives logic-derived information describing the current temporal requirement. The principal change is that the semantic object being embedded is different. We first describe how SemLTL embeds its LDBA labels and then show how the same feature families are adapted to the single-label DWA representation.

5.3.1 SemLTL$_f^+$: A Single DWA Semantic Label

In SemLTL, a state may be associated with two formulae, $\ell_{\text{SemLTL}}(q) = (\varphi_M(q), \varphi_B(q))$, where φ_M is the main formula and φ_B is the breakpoint formula when one is required. The breakpoint component is particularly relevant in the accepting component of an LDBA, where it tracks progress towards the next fulfilment of a recurring obligation [1]. SemLTL computes semantic features for the formulae separately and combines them to form the state embedding. The feature families are designed to approximate the behavioural meaning of a formula without requiring the policy network to manipulate its syntax tree directly. Two of the most important feature families are *trueness* and *propositional attention*. SemLTL additionally uses formula-complexity features, which are useful when the policy must compare alternative ϵ-successors [1].

SemLTL$_f^+$ does not retain the main/breakpoint decomposition. Since the DWA construction is deterministic and does not use the LDBA breakpoint mechanism to generate repeated Büchi acceptance, each DWA state is represented by a single semantic label,

$$\ell_+ : Q \to \mathcal{L}_{\text{sem}}, \qquad q \mapsto \ell_+(q).$$

The numerical task representation is computed directly as $z(q) = E_+(\ell_+(q))$. In SemLTL, feature extraction must represent both the main temporal requirement and, where present, the auxiliary breakpoint obligation. In SemLTL$_f^+$, the DWA state and its semantic annotation already represent the relevant obligation progress, so the feature extractor is applied once to the state label.

As a consequence, the semantic feature vector used in our implementation has dimension 167 rather than the 386-dimensional representation used by the corresponding SemLTL configuration. The reduction follows from removing duplicated main/breakpoint feature blocks and features whose purpose is tied specifically to resolving the LDBA structure. The resulting vector retains the logic-derived information used for task conditioning while removing components that have no analogue in the deterministic weak-automaton pipeline.

The remainder of this section explains the principal semantic features retained from SemLTL.

Embedding the Semantic Labels. The semantic labelling associated with an automaton state provides a symbolic description of the temporal obligations that remain to be satisfied. However, these labels cannot be supplied directly to a reinforcement learning policy: the policy requires a fixed-dimensional numerical representation that is comparable across different states and, importantly, across different task specifications. The objective of the embedding procedure is therefore to map semantic labels into a feature space in which states imposing similar future behaviours receive similar representations.

This requirement is particularly important in the multi-task setting. An automaton state identifier, such as (q_i), has no intrinsic meaning across independently generated automata; state (q_1) in one task need not bear any relationship to state (q_1) in another. In contrast, the LTL formulae attached to semantically labelled states explicitly describe their remaining temporal obligations. Embedding these formulae therefore provides a task-independent interface through which the policy can reason about progress.

A useful conceptual measure of similarity between two LTL formulae is the extent to which their languages overlap. Computing such language-level similarity directly, however, would generally require reasoning over infinite traces and is consequently unsuitable for an embedding that must be evaluated repeatedly during reinforcement learning. Instead, the semantic embedding approximates relevant aspects of the language by probing how individual propositions affect the formula. An effective embedding function must capture two crucial aspects of the residual task: (1) satisfaction status and distance, indicating how close subformulas are to completion or violation, and (2) relevance and attention, identifying which atomic propositions $\mathcal{AP}$ actively govern the immediate transition dynamics versus those that are dormant. Two complementary feature families are considered: trueness, which measures

the immediate effect of observing a proposition, and propositional attention, which captures dependencies between propositions and thereby provides information about future obligations.

For a semantically labelled automaton state containing both a main formula (φ_M) and, where applicable, a breakpoint formula (φ_B), the features described below can be computed independently for each formula and subsequently combined to obtain the representation of the complete semantic label.

5.3.2 Trueness and its Variations

The first component of the semantic embedding is based on *trueness* [24], a propositional approximation of how easily an LTL formula may be satisfied. The central idea is to temporarily abstract away the internal temporal semantics of temporal subformulae and instead treat those subformulae as independent Boolean variables. The proportion of assignments satisfying the resulting propositional formula then provides a coarse numerical indication of proximity to satisfaction.

Formally, let φ be an LTL formula and let $\psi_1, \ldots, \psi_n$ denote the relevant temporal subformulae abstracted during the propositionalisation procedure. Each ψ_i is replaced by a fresh Boolean variable x_{ψ_i}, producing a propositional abstraction $\text{prop}(\varphi)$. The trueness of φ is then defined as the fraction of Boolean valuations satisfying the abstraction,

$$\frac{|\{\nu \in \{0,1\}^n \mid \nu \models \widehat{\varphi}\}|}{2^n}$$

Consequently, $0 \leq \text{tr}(\varphi) \leq 1$. A larger value indicates that the propositional abstraction admits a greater fraction of possible assignments and can therefore be interpreted as being, in this approximate sense, easier to satisfy.

Trueness alone is a property of a formula rather than a description of how an observation affects that formula. For the UPI, the more useful quantity is therefore the change in trueness induced by formula progression. Given an observed letter $\sigma \in 2^{AP}$, let $\text{prog}(\varphi, \sigma)$ denote the progression of φ after observing σ. The trueness feature is defined as $\text{tr}(\varphi)$. This difference has a direct interpretation. A positive value indicates that observing σ moves the specification towards satisfaction; a negative value indicates that the observation makes satisfaction more difficult or violates an obligation; and a value of zero indicates that the observation has no detectable immediate effect under the propositional approximation.

For a set of environment propositions $AP = p_1, \ldots, p_m$, these quantities can be collected into a feature vector, for example, $\begin{bmatrix} f_{\text{tr}}(\varphi, p_1) & \cdots & f_{\text{tr}}(\varphi, p_m) \end{bmatrix}^\top$. The representation therefore gives the policy an explicit indication of which currently observable propositions are useful, irrelevant, or harmful with respect to the remaining temporal objective.

Example 5.3.1. *Consider the specification $\varphi = \mathbf{F}r \wedge \mathbf{FG}y$, which requires the proposition (r) to eventually occur and, independently, requires the system eventually to reach a point from which (y) holds permanently.*

To calculate its trueness, the temporal subformulae are propositionally abstracted. Let $x_1 = x_{\mathbf{F}r}, x_2 = x_{\mathbf{FG}y}$. The abstraction is therefore $\widehat{\varphi} = x_1 \wedge x_2$. There are four possible Boolean assignments.

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Exactly one of the four satisfies the formula. Hence, $\frac{1}{4} = 0.25$.

Suppose now that the agent observes (r). Progressing the formula through the letter (r) discharges the first eventuality, $\mathbf{FG}y$. The remaining temporal formula is represented by a single propositional variable, $\widehat{\varphi}' = x_{\mathbf{FG}y}$. One of its two possible assignments satisfies it, giving $\frac{1}{2} = 0.5$. The resulting trueness feature for observing (r) is therefore

$$\text{tr}(\mathbf{F}r \wedge \mathbf{FG}y) \; = 0.5 - 0.25 \quad = 0.25$$

The positive value reflects that observing (r) constitutes meaningful progress: one of the two outstanding obligations has been discharged. By contrast, observing (y) at this stage does not discharge the outstanding $\mathbf{F}r$ requirement and does not by itself establish the eventual-persistence property. Under progression, the relevant propositional structure is therefore unchanged, yielding $f_{\text{tr}}(\varphi, y) = 0$. Thus, even before reinforcement learning is applied, the embedding exposes a useful distinction between the propositions: (r) is immediately progress-making, whereas (y) does not presently alter the task's progress.

Trueness can similarly represent negative progress. Consider, for instance, $\varphi = \mathbf{G}y$. Its propositional abstraction contains a single temporal variable and hence has trueness $\text{tr}(\mathbf{G}y) = \frac{1}{2}$. If (y) is absent from the current observation, the safety requirement is violated and progression yields $\bot$. Since $\text{tr}(\bot) = 0$, the corresponding feature

is $0-\frac{1}{2} = -0.5$. The sign of the feature therefore distinguishes progress from violation, giving the policy substantially more information than an opaque automaton-state identifier.

Nevertheless, trueness deliberately sacrifices temporal information in exchange for computationally useful propositional structure. It should therefore be interpreted as an approximation rather than a satisfaction probability. In particular, temporal relationships can render a formula inconsistent even when its propositional abstraction is satisfiable. For example, $\mathbf{F}r \wedge \mathbf{G}\neg r$ is unsatisfiable: the first conjunct requires (r) eventually to occur, whereas the second prohibits (r) at every timestep. Its propositional abstraction nevertheless treats these temporal requirements as independent variables and assigns the resulting conjunction non-zero trueness. This limitation motivates a complementary representation capable of exposing dependencies that extend beyond the immediate progression step.

5.3.3 Propositional Attention

Although trueness identifies which propositions produce immediate progress, it is inherently myopic. It does not describe how satisfying one subgoal changes the relevance of other propositions in the future. Consider again $\mathbf{F}r \wedge \mathbf{FG}y$. The trueness feature correctly identifies (r) as immediately useful. However, knowing only that (r) increases trueness does not reveal an important structural property of the task: after (r) has been observed, (y) becomes the proposition governing the remaining objective. In an environment where the agent must choose how to reach (r), this distinction may affect the optimal action well before (r) is actually encountered. A purely immediate representation may consequently encourage locally sensible but globally poor decisions.

Propositional attention augments the embedding with precisely this information. Rather than assigning a score to a proposition independently, it considers ordered pairs $(p, q) \in AP \times AP$ and measures how relevant (q) becomes after (p) is observed. The construction first progresses the current formula with respect to (p), $\mathrm{prog}(\varphi, p)$. The future requirements of φ_p are then approximated through its *obligation sets* [26], $\mathrm{ob}(\varphi_p)$. Intuitively, an obligation set ($o \subseteq AP$) represents a propositional configuration which, if repeated indefinitely, is sufficient for satisfying the corresponding approximated temporal obligation. In particular, for $o \in \mathrm{ob}(\varphi_p)$, the infinite repetition o^ω satisfies the relevant progressed formula under the obligation-set construction. Obligation sets therefore provide a tractable propositional summary of longer-term behaviour rather than only the immediate effect captured by trueness.

The positive propositional attention from (p) to (q) is defined as

$$\frac{|\{o \in \mathrm{ob}(\varphi_p) \mid q \in o\}|}{|\mathrm{ob}(\varphi_p)|}$$

where $\mathrm{prog}(\varphi, p)$.

If the obligation-set collection is empty, the feature is conventionally assigned the value (0). Similarly, the negative propositional attention is

$$\frac{|\{o \in \mathrm{ob}(\varphi_p) \mid \neg q \in o\}|}{|\mathrm{ob}(\varphi_p)|}$$

The two quantities should be kept distinct: a future requirement for (q) and a future requirement for avoiding (q) have fundamentally different implications for the policy. For $m = |AP|$ atomic propositions, the positive attention features may be viewed as a matrix $A^+_\varphi \in [0,1]^{m \times m}$, with entries $f^+_{\mathrm{att}}(\varphi, p_i, p_j)$. An analogous matrix A^-_φ represents negative dependencies. These matrices may subsequently be flattened and concatenated with the trueness features to form part of the fixed-dimensional semantic embedding.

Importantly, the term attention here should not be conflated with learned self-attention in a Transformer. The values are not learned query-key similarities. Instead, they are logic-derived dependency features. The entry associated with (p, q) quantifies how frequently q occurs among the future obligation sets induced after observing p. The analogy to attention arises because the representation expresses which propositions should become relevant conditional on another proposition being encountered.

Example 5.3.2. *Consider once more* $\mathbf{F}r \wedge \mathbf{FG}y$. *Suppose that we wish to determine the positive propositional attention from (r) to (y).*

We first progress the formula after observing (r).

$$\varphi_r = \mathrm{prog}(\mathbf{F}r \wedge \mathbf{FG}y, r) \quad = \mathbf{FG}y.$$

The remaining formula requires the system eventually to enter a regime in which (y) holds indefinitely. At the propositional level, its relevant long-term obligation therefore contains (y). In the simple case, $\{y\}$. There is one obligation set in total, and y occurs in that set. Hence,

$$f^+_{\mathrm{att}}(\varphi, r, y) = \frac{|o \in \mathrm{ob}(\varphi_r) \in o|}{|\,\mathrm{ob}(\varphi_r)|} \quad = \frac{1}{1} \; = 1.$$

This value encodes a strong dependency: once (r) has been achieved, (y) is central to satisfying the remaining task. Now consider the self-dependency from (r) to (r). After r has occurred, the eventuality $\mathbf{F}r$ has already been discharged. Consequently, r does not occur in the remaining obligation set, $r \notin y$. Thus, $\frac{0}{1} = 0$. The pair of values $f^{+}_{\text{att}}(\varphi, r, r) = 0, \qquad f^{+}_{\text{att}}(\varphi, r, y) = 1$ provides information that cannot be recovered from the immediate trueness feature alone. Together, they tell the policy that r is currently useful, but that once r has been achieved, further occurrences of r are irrelevant while y becomes essential.

More generally, suppose a progressed formula admits four obligation sets.

$$\{\{q, r\}, \{q, s\}, \{r, s\}, \{q, r, s\}\}$$

The proposition q appears in three of the four sets. Therefore, $\frac{3}{4} = 0.75$. The resulting value does not state that q is logically necessary in every satisfying execution. Rather, it expresses that q appears in a large fraction of the propositional approximations of the future obligations induced by observing p. Propositional attention is intended as a dense semantic feature for representation learning, not as an exact decision procedure for LTL satisfaction.

Trueness and propositional attention therefore provide complementary views of an automaton state's semantics. Trueness answers the local question *"What effect would observing (p) have on the task now?"* whereas propositional attention addresses the conditional, longer-term question *"If (p) is observed now, which propositions become relevant afterwards?"* The former supplies an explicit signal of immediate progress or violation, while the latter exposes dependencies among sequential subgoals. Their combination consequently preserves both local progress information and a coarse representation of future temporal structure. When applied to the formulae constituting the semantic label of an automaton state, these features transform symbolic temporal obligations into a fixed-dimensional numerical interface suitable for conditioning a universal reinforcement learning policy.

5.4 RL Pipeline: Removing ϵ-Actions

The final architectural change concerns how the automaton interacts with the reinforcement learning policy. This change follows directly from replacing an LDBA with a DWA.

5.4.1 Learnable ϵ-Actions

SemLTL's LDBA may contain ϵ-transitions from its initial component into its deterministic accepting component. These transitions do not consume an environment observation and do not change the physical MDP state. Nevertheless, choosing among them can affect whether the resulting automaton run is accepting. SemLTL therefore exposes available ϵ-transitions as choices to the policy [1].

For current LDBA state q, let $\mathcal{E}(q)$ be the available ϵ-successors. The effective action set contains both physical environment actions and automaton actions:

$$A_{\text{SemLTL}}(q) = A \cup \{\epsilon_u \mid u \in \mathcal{E}(q)\}.$$

SemLTL uses a multi-headed policy architecture to handle this hybrid space. The semantic embeddings of the current state and its ϵ-successors are projected into latent representations. A scoring head determines whether an ϵ-transition should be taken, while an environment actor head produces a distribution over the physical action space. When no ϵ-transition is available, the policy reduces to the environment actor [1]. The semantic representation therefore serves two roles in the original architecture: it helps choose an appropriate physical action and helps resolve the automaton nondeterminism introduced by the LDBA construction.

5.4.2 SemLTL$_f^+$: Environment Actions Only

The DWA used by SemLTL$_f^+$ is deterministic. Once the environment has produced the next propositional valuation, automaton progression is uniquely determined:

$$q_{t+1} = \delta\left(q_t, L(s_{t+1})\right).$$

There is consequently no automaton decision for the agent to make. SemLTL$_f^+$ removes the ϵ-action scoring head and exposes only the underlying MDP action space,

$$A_{\text{SemLTL}_f^+} = A.$$

At timestep t, the semantic feature vector of the current DWA state is projected and combined with the encoded physical state. The policy therefore has the standard task-conditioned form

$$a_t \sim \pi_\theta\left(\cdot \mid s_t, z_t\right), \qquad z_t = E_{\text{SemLTL}_f^+}(q_t).$$

After executing a_t, the environment transitions to s_{t+1}. Its propositional label is evaluated and used internally to advance the DWA. The policy does not choose this transition:

$$s_t \xrightarrow{a_t} s_{t+1} \xrightarrow{L} \sigma_{t+1} \xrightarrow{\delta} q_{t+1} \xrightarrow{E} z_{t+1}.$$

This produces a cleaner separation between control and monitoring. The policy controls the environment, while the DWA deterministically records the temporal consequences of the resulting trajectory.

5.4.3 Consequences for the Universal Progress Interface

The Universal Progress Interface introduced in the preceding chapter can now be instantiated directly. SemLTL$_f^+$ maintains the DWA state as symbolic task memory and exposes its semantic embedding as the policy-facing representation:

$$\rho_t = q_t, \qquad z_t = E_{\text{SemLTL}_f^+}(q_t).$$

The update function is

$$\mathcal{U}\left(q_t, \sigma_{t+1}\right) = \delta\left(q_t, \sigma_{t+1}\right).$$

Thus, the complete interaction loop is

$$\boxed{s_t} \longrightarrow \boxed{\pi_\theta(\cdot \mid s_t, z_t)} \longrightarrow \boxed{a_t} \longrightarrow \boxed{s_{t+1}} \longrightarrow \boxed{L(s_{t+1})} \longrightarrow \boxed{q_{t+1}} \longrightarrow \boxed{z_{t+1}}.$$

The automaton and embedding continue to play different roles. The DWA is responsible for maintaining correct temporal memory; the 167-dimensional semantic embedding is responsible for presenting that progress in a form that can be shared across tasks. This separation is particularly important in MTRL: correctness of temporal monitoring does not depend on what the neural network learns, while transfer between specifications does not depend on arbitrary DWA state identifiers.

5.5 End-to-End SemLTL$_f^+$ Framework

The complete SemLTL$_f^+$ framework can be viewed as a modification of SemLTL at three interfaces.

1. Specification to monitor. SemLTL maps a full LTL formula to a semantically labelled LDBA. SemLTL$_f^+$ maps an LTL$_f^+$ obligation to a DWA constructed from its finite-trace components.

2. Monitor state to task embedding. SemLTL embeds semantic information consisting of main and breakpoint formulae. SemLTL$_f^+$ embeds a single semantic DWA label and reduces the task representation from 386 to 167 dimensions.

3. Task embedding to control. SemLTL uses the embedding both to select environment actions and to resolve available LDBA ϵ-transitions. SemLTL$_f^+$ removes the latter responsibility because DWA progression is deterministic; the policy outputs environment actions only. The complete pipeline is therefore

$$\Psi \xrightarrow{\text{obligation construction}} \mathcal{W}_\Psi \xrightarrow{\text{execution}} q_t \xrightarrow{\text{semantic labelling}} \ell_+(q_t)$$
$$\xrightarrow{\text{feature extraction}} z_t \in \mathbb{R}^{167} \xrightarrow{\text{policy conditioning}} \pi_\theta(\cdot \mid s_t, z_t).$$

This formulation makes explicit which aspects of SemLTL are inherited and which are changed. The inherited principle is *semantic task conditioning*: automata maintain exact temporal progress while logic-derived features expose the meaning of that progress to a universal policy. The changes are induced by the structure of LTL$_f^+$ obligations: deterministic weak automata replace LDBAs, a single DWA semantic label replaces the main/breakpoint representation, and deterministic automaton progression removes ϵ-actions from the reinforcement-learning problem. The flow of control for different parts of the framework can also be observed in the original SemLTL architecture overview in Figure 5.1.

This chapter introduced SemLTL$_f^+$ by systematically adapting the components of SemLTL to LTL$_f^+$ obligations. The original SemLTL framework demonstrates that semantically labelled automata can simultaneously provide task memory and a transferable task representation for multi-task reinforcement learning [1]. SemLTL$_f^+$ retains this separation between symbolic monitoring and policy-facing semantic representation but changes the automata-theoretic machinery underneath it.

At the automaton level, the LDBA is replaced by a deterministic weak automaton derived from the LTL$_f^+$ obligation construction. This removes the restricted automaton nondeterminism and ϵ-transitions present in SemLTL. At the representation level, the main/breakpoint formula decomposition is replaced by a single semantic DWA

label. Trueness and propositional attention remain the principal mechanisms for converting logical information into numerical features, but the resulting representation is reduced from 386 to 167 dimensions. Finally, at the reinforcement-learning level, the policy no longer chooses between physical actions and automaton ϵ-actions: it acts only in the environment, while the DWA progresses deterministically from the observed propositional labels. The resulting architecture preserves the main representational advantage of SemLTL while aligning the monitor, embedding, and policy interface with the structure of LTL_f^+ obligations. The following chapter evaluates whether these changes preserve effective cross-task generalisation and examines the computational and learning behaviour of the resulting SemLTL_f^+ agent.

Chapter 6

Experimental Evaluation

The preceding chapters established the theoretical foundations of multi-task reinforcement learning under non-Markovian temporal constraints, culminating in the formal integration of Semantically Labelled Automata as an Universal Progression Interface (UPI). While theoretical guarantees ensure the expressiveness and structural correctness of our automaton-guided framework, validating its practical efficacy requires rigorous empirical evaluation across complex, high-dimensional decision-making environments. This chapter details a practical implementation of our proposed framework and presents a comprehensive experimental suite designed to evaluate the performance, sample efficiency, and generalization capabilities of our proposed approach. We describe the software implementation and experimental infrastructure used to evaluate the proposed LTL_f^+-obligation extension of SemLTL. The logical and automata-theoretic construction itself was presented in the preceding chapter; the focus here is instead on how this construction is integrated into the reinforcement-learning codebase, the environments used for evaluation, and the training infrastructure. We benchmark our framework against the state-of-the-art baseline, SemLTL, across diverse task specifications. Specifically, our empirical investigation is structured around two core research questions.

1. Sample Efficiency and Task Convergence: Does adapting semantically labelled automata for LTLf+ Obligations using DWA accelerate training convergence and success rate compared to an LDBA based methodology?

2. Zero-Shot Task Generalization: How effectively does the Universal Progression Interface execute completely unseen LTL instructions at test time without requiring policy fine-tuning or parameter updates?

6.1 Implementation

The implementation is developed as an extension of the SemLTL[1] experimental codebase of Abate et al. [1]. The original codebase builds on the experimental infrastructure developed for DeepLTL [22], and the present implementation *mostly* preserves this overall reinforcement-learning and environment stack while replacing the temporal-logic backend required for task monitoring and semantic embedding. The main reinforcement-learning implementation is written in **Python** while the automaton formulations are implemented in **Java**. All experiments are executed through the same training and evaluation infrastructure, so differences between the original SemLTL agent and the proposed LTL_f^+ variant arise from the task-generation, automaton and semantic-representation components rather than from different RL implementations. The implementation can therefore be viewed as consisting of three software layers.

1. an *environment layer*, responsible for simulating the MDP and returning observations and proposition labels;
2. a *temporal-task layer*, responsible for sampling specifications, constructing and updating their automata, and producing semantic task embeddings; and
3. an *RL layer*, responsible for encoding environment observations and learning the universal actor–critic policy.

6.1.1 Automaton Construction

The automaton construction package, FishSemML, was implemented in Java. The original SemLTL pipeline uses SemML to translate LTL formulae into semantically labelled LDBAs. The proposed pipeline uses FishSemML to translate LTLf+ obligations into deterministic weak Büchi automata. Both tools produce automata in the Hanoi Omega Automata format (HOA), which provides a common representation for automaton states, labelled transitions and Büchi acceptance conditions. FishSemML was compiled as an executable Java archive. It was invoked from Python using a subprocess of the following general form.

```
java -jar fsemml-dev-all.jar embed \
--formula <formula> \
--aps <atomic-propositions> \
```

[1] https://github.com/mathiasj33/semltl/tree/obligations_exp

```
    --eligible-letters <environment-letters> \
    --output <output-file.hoa>
```

A temporary output file was created for each invocation. The resulting HOA representation is read into Python and the temporary file was subsequently deleted. Standard output and standard error from the Java process were captured so that invalid formulae and automaton-generation failures could be reported through the Python training pipeline.

6.1.2 Reinforcement Learning

The reinforcement-learning pipeline was implemented in Python 3.12 using **PyTorch** [30] and JAX for accelerated numerical computation. Environments follow the **Gymnasium** interface [25]. Equinox was used to define and serialise the neural-network models, while Hydra was used to manage experiment configurations and command-line overrides. Training environments were vectorised and compiled using JAX just-in-time compilation. The original SemML representation constructs a 386-dimensional semantic vector for each automaton state. For a state in the initial LDBA component, this vector is formed by concatenating its formula embedding with a zero-valued breakpoint embedding. For a state in the accepting component, the master-formula and breakpoint-formula embeddings are concatenated.

FishSemML directly provides a single `formula_embedding` for each DWA state. In the current ZoneEnv8 configuration, this embedding is an array of length 167. The proposed representation reduces the semantic input dimensionality from 386 to 167 (56.7%) and reduces the complete policy parameter count from 59.37K to 52.36K (11.8%). The parsed automata are converted into batched JAX arrays before training. The representation contains: (1) The number of states, (2) The initial state, (3) A deterministic transition table, (4) Büchi transition-acceptance indicators, (5) State embeddings, (6) Epsilon-transition arrays, (7) Sink-state indicators, (8) Accepting-state metadata, (9) Accepting bottom-component indicators, and (10) Rejecting bottom-component indicators. The epsilon-transition array is retained to preserve compatibility with the existing SemLTL observation and model structures. However, for DWAs, it contains **no enabled epsilon transitions**.

FishSemML state metadata is used to classify bottom strongly connected components. A bottom component is accepting when every state in the component is accepting, and rejecting when none of its states is accepting. Entry into either type of component permanently determines the Büchi result because no transition can leave

a bottom component. Additional handling was introduced for states that become unreachable after pruning. Such states may remain in the parsed HOA representation, but do not belong to a reachable strongly connected component. These states are ignored when classifying reachable bottom components. A dedicated semantic DWA wrapper was introduced to interpret DWA states and rewards appropriately. Especially, this wrapper handles sink states as DWAs are guaranteed to have sink states for every strongly connected component. This helps speed up the training process by stopping an episode training early on hitting a rejecting sink and vice versa with an accepting sink.

The observation retains the same general structure as original SemLTL: physical environment features + current automaton-state embedding + epsilon-transition embeddings + epsilon-transition mask. However, all epsilon embeddings are zero and all epsilon-mask entries are false. A DWA is deterministic and therefore has **no nondeterministic epsilon choices for the policy to resolve**. The actor still contains the epsilon-related output inherited from SemLTL. The DWA wrapper ignores this component of the action. Retaining it allowed the existing SemLTL model architecture to be reused, although a future DWA-specific architecture could remove this unused output entirely.

6.2 Experimental Setup

The experiments compare two agents:

1. Original SemLTL: standard LTL instructions translated into semantically labelled LDBAs using SemML.

2. Guarantee-oriented Obligation model: LTLf+ obligation instructions translated into semantically labelled DWAs using FishSemML, trained with constrained data.

3. Obligation model: LTLf+ obligation instructions translated into semantically labelled DWAs using FishSemML.

Both agents use the same ZoneEnv8 environment and PPO implementation. They differ primarily in the logical language, automaton representation, semantic embeddings and handling of automautomaton transitions. Using the same RL algorithm and environment isolates the effect of the automaton and formula representation as much as possible.

The SemLTL$_f^+$ G agent represents the original guarantee-oriented version of the proposed approach. It was trained in ZoneEnv8 using a seven-stage curriculum that progressively introduced existential reach, sequential reach, reach-avoid, sequential reach-avoid, existential weak-next, universal weak-next response, and general disjunctive reach-avoid specifications. Representative formulas included $E(Fp)$, $E(F(p \wedge Fq))$, $E(\neg q\ U\ p)$, and $E(\neg(q \vee r)\ U\ (p \vee s))$. Each curriculum stage primarily contained the newly introduced task family, and advancement was controlled by a performance threshold calculated over a moving window of 128 completed episodes. Importantly, this curriculum did not contain standalone safety formulas, conjunctive guarantees, or conjunctions of safety and guarantee obligations. Performance on these families therefore measured generalisation to structurally unseen specification types.

The revised SemLTL$_f^+$ agent used the same ZoneEnv8 environment, FishSemML automaton pipeline, neural architecture, and PPO implementation, but employed a broader matched-mixture curriculum. The early stages introduced simple reach, sequential reach, reach-avoid, nested reach-avoid, Boolean or disjunctive reach-avoid, standalone conjunctive guarantees, and pure safety tasks. Earlier task families were replayed in later stages to reduce catastrophic forgetting. The final stage sampled 50% broad reach or reach-avoid obligations, 25% pure safety obligations, and 25% combined safety-and-guarantee obligations. The broad reach-avoid sampler varied temporal depth between one and two, selected between one and two alternative goal propositions, and selected between zero and two avoidance propositions. It therefore produced formulas with multiple goals, multiple hazards, disjunctions, and nested temporal structure. The final 25% safety-and-guarantee allocation was divided equally between specifications containing one guarantee and one safety condition and specifications containing multiple guarantees and multiple safety conditions. Examples were $E(Fp) \wedge A(G\neg q)$ and $E(Fp \wedge Fq) \wedge A(G\neg(r \vee s))$. In contrast to SemLTL$_f^+$ G, pure safety, conjunctive guarantees, general disjunctive reach-avoid, and safety-plus-guarantee specifications were therefore structurally represented during training. Evaluation on these categories measures generalisation to unseen proposition combinations and formula instances, rather than generalisation to entirely unseen specification families.

Both agents received semantic automata generated by the Java FishSemML implementation and represented in HOA format. The automata were parsed and batched by the Python/JAX training pipeline. FishSemML supplied a 167-dimensional semantic feature vector for each automaton state, which was projected by the policy network into a learned 32-dimensional representation and combined with the agent's

visual and proprioceptive environment features. Both systems used deterministic weak automata, so automaton progression was determined by the propositions observed after each physical environment action. The revised SemLTL$_f^+$ implementation additionally removed the unused epsilon-action policy head. Its actor returned only a continuous physical action because the deterministic weak automata contain no epsilon transitions. In the earlier implementation, epsilon transitions were disabled through a mask, but the shared SemLTL actor still instantiated the corresponding policy head and always selected its non-epsilon "stay" option. Removing this component made the revised architecture consistent with the deterministic automaton semantics and reduced unnecessary computation, although it was not expected to produce a large change in task performance because epsilon actions had already been masked.

The central experimental difference between the two agents is therefore their training distribution. SemLTL$_f^+$ G tested whether a guarantee-oriented curriculum could generalise to safety and mixed obligation types that it had never encountered. The revised SemLTL$_f^+$ agent instead tested performance after aligning the curriculum with the full range of target specification families and continually rehearsing earlier tasks. Comparing them quantifies how much of the original performance limitation resulted from the automaton representation itself and how much resulted from incomplete curriculum coverage or catastrophic forgetting.

6.2.1 Expressive Equivalence of LTL$_f^+$ and LTL

Although we compare two different frameworks based on two different logical languages, we are able to do so due to an existing equivalence between them.

Although LTL$_f^+$ is defined by quantifying over finite prefixes, the resulting formulae describe properties of *infinite* traces. Consequently, LTL$_f^+$ can be compared directly with standard Linear Temporal Logic (LTL), which is also interpreted over infinite traces. A central result of Aminof et al. [3] is that the two logics have the same expressive power.

Let $[\![\Psi]\!] \subseteq (2^{\mathcal{AP}})^\omega$ denote the infinite-trace language defined by an LTL$_f^+$ formula Ψ, and let $\mathcal{L}(\varphi)$ denote the infinite-trace language of an LTL formula φ. Then the expressive-equivalence result can be stated as follows.

Theorem 6.2.1 (Expressive Equivalence [3])**.** LTL$_f^+$ *and LTL define exactly the same class of infinite-trace properties. In particular,*

$$\forall \Psi \in \mathrm{LTL}_f^+, \quad \exists \varphi \in \mathrm{LTL} \quad \textit{such that} \quad [\![\Psi]\!] = \mathcal{L}(\varphi),$$

and conversely,

$$\forall \varphi \in \mathrm{LTL}, \quad \exists \Psi \in \mathrm{LTL}_f^+ \quad \textit{such that} \quad \mathcal{L}(\varphi) = [\![\Psi]\!].$$

Thus, moving from LTL to LTL_f^+ does not restrict which ω-regular properties expressible in LTL can ultimately be specified. Instead, the two logics provide different *representations* of these properties. LTL specifies an infinite execution directly using temporal operators, whereas LTL_f^+ describes the same class of infinite-trace behaviours by reasoning about which finite prefixes satisfy an underlying LTL_f property.

Prefix interpretation. Let $\tau = \sigma_0\sigma_1\sigma_2\cdots \in (2^{\mathcal{AP}})^\omega$ be an infinite trace and let $\tau_{\leq i} = \sigma_0\sigma_1\cdots\sigma_i$ denote its non-empty finite prefix ending at position i. For an LTL_f formula Φ, the four primitive LTL_f^+ prefix quantifiers are interpreted as

$$\tau \models \forall\Phi \iff \forall i \geq 0 : \tau_{\leq i} \models_f \Phi, \tag{6.1}$$

$$\tau \models \exists\Phi \iff \exists i \geq 0 : \tau_{\leq i} \models_f \Phi, \tag{6.2}$$

$$\tau \models \forall\exists\Phi \iff \forall i \geq 0\ \exists j \geq i : \tau_{\leq j} \models_f \Phi, \tag{6.3}$$

$$\tau \models \exists\forall\Phi \iff \exists i \geq 0\ \forall j \geq i : \tau_{\leq j} \models_f \Phi. \tag{6.4}$$

These correspond respectively to the *safety*, *guarantee*, *recurrence*, and *persistence* levels of the safety–progress hierarchy of Manna and Pnueli [27]. Arbitrary Boolean combinations of these properties yield the reactivity level. The safety–progress hierarchy is central to the expressive-equivalence result: reactivity properties capture the same infinite-trace properties as LTL [27, 3].

Why the equivalence holds. The relationship can be understood through Pure-Past LTL (PPLTL). LTL_f and PPLTL define the same class of finite-trace properties, namely the first-order-definable, or equivalently star-free regular, languages over finite traces [10, 3]. Consequently, an LTL_f property appearing below a prefix quantifier can be replaced, at the level of expressiveness, by an equivalent PPLTL property.

Manna and Pnueli's hierarchy then establishes that Boolean combinations of the corresponding recurrence properties are sufficient to express arbitrary LTL properties over infinite traces. Combining these results gives

$$\mathrm{LTL}_f \equiv_{\mathrm{fin}} \mathrm{PPLTL} \quad \Longrightarrow \quad \mathrm{LTL}_f^+ \equiv_\omega \mathrm{PPLTL}^+ \equiv_\omega \mathrm{LTL},$$

where $\equiv_{\mathrm{fin}}$ denotes equality of expressive power over finite traces and $\equiv_\omega$ denotes equality over infinite traces [3].

Importantly, expressive equivalence should not be interpreted as syntactic equivalence. In general, converting an arbitrary LTL_f^+ formula to LTL is not achieved simply by replacing the prefix quantifiers $\forall$, $\exists$, $\forall\exists$, and $\exists\forall$ with $\mathbf{G}$ and $\mathbf{F}$. The finite-trace formula beneath the quantifier is evaluated over an *entire prefix*, whereas an LTL subformula is evaluated at a position of an infinite trace. The correspondence is therefore semantic, and arbitrary translations may incur a substantial change in formula representation. For simple properties, however, the correspondence between LTL_f^+ and LTL can be seen directly.

Example 6.2.1 (Safety)**.** *Consider the requirement that a hazardous state h must never occur. Let*

$$\Phi_{\mathrm{safe}} = \mathbf{G}\neg h$$

be interpreted under finite-trace semantics. The LTL_f^+ *formula*

$$\Psi_{\mathrm{safe}} = \forall\Phi_{\mathrm{safe}} = \forall(\mathbf{G}\neg h)$$

requires every finite prefix *of the infinite execution to satisfy $\mathbf{G}\neg h$. This holds exactly when h never occurs anywhere in the infinite execution. Hence,*

$$\boxed{\forall(\mathbf{G}\neg h) \equiv \mathbf{G}\neg h}$$

where the formula on the right is interpreted using ordinary infinite-trace LTL semantics.

Example 6.2.2 (Guarantee)**.** *Suppose that proposition d represents successful delivery. Consider the finite-trace property*

$$\Phi_{\mathrm{deliver}} = \mathbf{F}d.$$

Then

$$\Psi_{\mathrm{deliver}} = \exists(\mathbf{F}d)$$

requires the existence of some finite prefix in which d has occurred. Such a prefix exists exactly when d eventually occurs on the infinite trace. Therefore,

$$\boxed{\exists(\mathbf{F}d) \equiv \mathbf{F}d}.$$

A sequential finite-trace objective behaves similarly. For

$$\Phi = \mathbf{F}\left(k \wedge \mathbf{F}d\right),$$

the LTL_f^+ *guarantee* $\exists\Phi$ *states that there must eventually exist a prefix witnessing a position at which k holds followed by a later position satisfying d. Its corresponding LTL property is*

$$\boxed{\exists\mathbf{F}\left(k \wedge \mathbf{F}d\right) \equiv \mathbf{F}\left(k \wedge \mathbf{F}d\right)}.$$

Example 6.2.3 (Safety + Guarantee)**.** *The obligation fragment considered in this thesis permits positive Boolean combinations of safety and guarantee properties. For example,*

$$\Psi = \forall(\mathbf{G}\neg h) \wedge \exists\mathbf{F}\left(k \wedge \mathbf{F}d\right)$$

requires the agent to remain safe on every prefix while eventually producing a prefix witnessing the ordered achievement of k followed by d. The equivalent LTL specification is

$$\boxed{\mathbf{G}\neg h \wedge \mathbf{F}\left(k \wedge \mathbf{F}d\right)}.$$

This example illustrates why the obligation fragment remains particularly natural for reinforcement-learning tasks: safety constraints and finite-prefix achievements can be expressed separately and subsequently combined using Boolean structure.

The same equivalence is established for recurrence and persistence formulas which we don't cover as it is not particularly relevant to our work. Similarly, LTL can also be converted to LTL_f^+ [47].

6.2.2 Environment & Learning

Experiments are conducted in `ZoneEnv8`, which contains eight semantically labelled coloured zones: blue, orange, green, red, purple, brown, pink, gray. The agent receives the existing ZoneEnv8 sensory observation, including the environment's visual, range and proprioceptive features. The semantic automaton-state representation is concatenated to the encoded environment observation before being processed by the actor and critic. An episode has a maximum duration of 1,000 environment steps. Episodes may end earlier when: the environment terminates naturally, an accepting bottom automaton component is reached, or a rejecting bottom automaton component is reached. Training and evaluation use separate sets of precomputed environment resets. This prevents evaluation from using exactly the same initial environment configurations as training.

Both models are trained using Proximal Policy Optimization (PPO) with an actor–critic architecture. No alternative RL algorithm is used in the main comparison. Table 6.1 lists the hyperparameters used in all of our experiments. Rewards are normalised during training using the same reward-normalisation wrapper for both approaches. The physical environment observation is processed using the existing SemLTL RGB/LiDAR convolutional network. It uses two convolutional stages with 16 and 32 channels and a kernel size of 5. The fused environment representation has width 64. The actor contains three hidden layers: $64 \to 64 \to 64$ with ReLU activations and a hyperbolic-tangent output activation for continuous physical actions. A state-dependent standard deviation is used for the stochastic policy. The critic contains two hidden layers: $64 \to 64$ with hyperbolic-tangent activations. The automaton embedding is projected to an internal semantic representation of width 32. While SemLTL uses a 386 length feature vector, SemLTL$_f^+$ uses only a 167 length feature vector.

Five independent training seeds are used. JAX pseudo-random keys are constructed independently for every seed. Separate keys are then used for model initialisation, environment sampling and policy-action sampling. Each seed therefore corresponds to an independently initialised and trained policy. The five models are trained in parallel using JAX vectorisation. Reported evaluation results should aggregate performance across these five independently trained models. Means should be accompanied by standard deviations across seeds rather than treating all evaluation episodes as independent training repetitions. Since each seed receives 3×10^8 environment steps, the complete five-seed experiment executes up to 1.5×10^9 steps across all independently trained agents.

Table 6.1: PPO training hyperparameters.

Parameter	**Value**
Training timesteps	3×10^8 per seed
Parallel environments	128
Rollout length	1,024 steps
Minibatches	4
Update epochs	5
Discount factor γ	0.998
GAE parameter λ	0.95
PPO clipping coefficient	0.2
Entropy coefficient	0.003
Value-loss coefficient	0.5
Learning rate	5×10^{-4}
Maximum gradient norm	1.0
Adam epsilon	10^{-8}
Learning-rate annealing	Disabled

6.2.3 Curriculum & Task Specification Types

Training uses a staged curriculum. Progression is tracked independently within the vectorised environments using a rolling window of 128 completed episodes. A stage is advanced when average performance over the window reaches the configured threshold.

Guarantee Formulae

Existential Reachability. These tasks constitute the simplest class of guarantee specifications and require the agent to eventually reach a target proposition. They have a single-step reachability structure and are represented by formulae of the form $\exists(F(p))$. For example, the agent must eventually reach a state satisfying proposition p. This class corresponds to Stage 1 of the curriculum, with an advancement threshold of 0.90.

Sequential Existential Reachability. These tasks extend basic reachability by requiring the agent to satisfy a sequence of reachability objectives. In particular, the agent must first reach a state satisfying p and subsequently reach a state satisfying q. The corresponding formulae have the form $\exists(F(p \wedge F(q)))$. This class corresponds to Stage 2, with an advancement threshold of 0.95.

Reach-Avoid. These tasks combine a reachability objective with an avoidance constraint. The agent must eventually reach a target proposition p while avoiding proposition q until the target is reached. This is expressed using the until operator as $\exists((\neg q), U, p)$. These tasks form Stage 3 of the curriculum and use an advancement threshold of 0.95.

Sequential Reach-Avoid. These tasks combine sequential reachability with multiple avoidance constraints. The agent must first reach p while avoiding q, and then satisfy a subsequent reach-avoid objective requiring s to be reached while avoiding r. These specifications are represented by $\exists((\neg q), U, (p \wedge ((\neg r), U, s)))$. This class corresponds to Stage 4 and uses an advancement threshold of 0.90.

General Reach-Avoid. The final curriculum stage consists of randomly sampled existential reachability and reach-avoid specifications. The sampled formulae have sequence depths between one and two, contain between one and two target propositions, and include between zero and two avoidance propositions. No advancement threshold is assigned to this stage, as it serves as the final generalization stage of the curriculum.

Existential Weak-Next Sequence. These tasks introduce the weak-next operator into an existential reachability specification. The agent must eventually reach a state satisfying p for which the subsequent state satisfies the weak-next condition on q. The corresponding formula is $\exists(F(p \wedge N(q)))$. This class constitutes Stage 5 of the curriculum, with an advancement threshold of 0.90.

Conjunctive Guarantee. These tasks require the agent to satisfy multiple reachability objectives within the same execution. The formula $E(F(\mathit{purple}) \wedge F(\mathit{brown}))$ requires that the agent eventually reach a state satisfying *purple* and, potentially at a different point in the execution, eventually reach a state satisfying *brown*. Unlike sequential reachability, the formula does not impose an ordering between the two objectives; it only requires that both targets are eventually reached.

Safety Formulae

Universal Weak-Next Response. These tasks introduce universal response requirements using the weak-next operator. The specification requires that whenever proposition p holds, the corresponding weak-next condition on q must also hold.

These tasks are represented by $A(p \rightarrow N(q))$. This class corresponds to Stage 6 and uses an advancement threshold of 0.90.

Pure Safety. These tasks specify a universal safety requirement in which the agent must avoid a designated proposition throughout the entire execution. In particular, the formula $A(G(\neg orange))$ requires that proposition *orange* never becomes true along any execution satisfying the specification. This represents a global avoidance constraint and therefore belongs to the safety class of specifications.

Safety & Guarantee Formulae

These tasks combine an existential reachability objective with a universal safety constraint. The formula $E(F(purple)) \wedge A(G(\neg orange))$ requires the agent to eventually reach a state satisfying *purple* while simultaneously ensuring that *orange* is never reached. This specification therefore combines a guarantee objective, represented by existential reachability, with a global safety requirement.

Each curriculum stage is precomputed prior to training. For each stage, the configuration requests 10,000 sampled formula–automaton instances. Precomputation avoids constructing Java-based automata within the JAX training loop, thereby separating the computationally expensive automaton-generation process from the training procedure. Training formulae are generated randomly from curriculum templates. Evaluation instead uses a fixed held-out set so that every trained model is tested on exactly the same instructions. The obligation evaluation set contains 20 formulae covering: Existential reachability, Sequential reachability, Reach-avoid, Sequential reach-avoid, General multi-target and multi-avoid instructions, pure safety formulae, unordered guarantees, and most importantly, combined safety-guarantee (obligation) formulae. Some of these task types might have been unseen by our model during training.

6.3 Results

We report the success rate for our methods on the evaluation formulae. For finite horizon tasks, a success rate is given along with average length of an episode. For infinite horizon tasks, the number of times the automaton visited in an episode versus

the length of the episode is recorded. The results in Tables 6.2 and 6.3 [2] provide an initial evaluation of SemLTL$_f^+$ and, more broadly, of semantically conditioned multi-task reinforcement learning for LTL$_f^+$ obligations. To the best of our knowledge, there is currently no established LTL$_f^+$ multi-task reinforcement-learning baseline directly comparable to the setting considered here. SemLTL therefore serves as the closest methodological reference point: it follows the same principle of conditioning a universal policy on semantic automaton representations, but operates over LTL using LDBAs rather than the DWA-based LTL$_f^+$ construction proposed in this work. The comparison should consequently be interpreted primarily as an investigation of whether the semantic-automata paradigm transfers successfully to this new setting, rather than as a controlled comparison between otherwise identical algorithms. The experiments therefore provide an initial assessment of the feasibility, learning behaviour, and generalisation potential of the proposed framework; a capacity-matched comparison and training over a broader obligation distribution are left for subsequent investigation.

6.3.1 Preliminary Results

Table 6.2 presents preliminary results for the SemLTL$_f^+$ model we initially trained with guarantee formulae. We call it SemLTL$_f^+$ G. Importantly, these results do not correspond to the final training configuration: in this version, the model has been trained only on *guarantee* formulae. The unseen task families are therefore included primarily to examine whether the learned semantic representation exhibits any transfer beyond the temporal structures encountered during training.

On the seen guarantee tasks, SemLTL$_f^+$ G performs comparably to SemLTL across all task families. Success rates are identical for Existential Reach and Sequential Reach, while SemLTL$_f^+$ G slightly improves performance on Reach-Avoid and General Reach-Avoid. The latter shows the largest improvement, with success rate increasing from 0.87 to 0.92 while the mean successful episode length decreases from 203 to 171 steps. Conversely, SemLTL performs slightly better on Sequential Reach-Avoid. Overall, these results suggest that replacing the LDBA-based representation with the smaller DWA-based semantic representation **does not substantially degrade performance on the guarantee tasks represented in the**

[2] To be noted that these numbers are rounded off. Hence 1.00 does not mean it averaged a 100% accuracy across runs.

training distribution, *even with a model with half the feature size*[3]. As expected, this model trains faster on a larger amount of data compared to the baseline.

The unseen results are particularly informative given the restricted training regime. SemLTL$_f^+$ G generalises almost perfectly to Conjunctive Guarantee, achieving a success rate of 0.99, despite this task structure not being encountered during training. More interestingly, the model achieves a success rate of 1.00 on Pure Safety. This task lies outside the guarantee-only training distribution and does not contain an eventual goal towards which the agent can explicitly progress. Its successful execution therefore provides preliminary evidence that the semantic representation captures information useful for avoidance behaviour even before safety formulae are explicitly included in training.

Performance is weaker on the more difficult unseen Safety + Guarantee composition, where SemLTL$_f^+$ G achieves 0.40 compared with 0.61 for SemLTL. This is not unexpected at the current stage: the model has not yet been trained on safety constraints or their composition with guarantee objectives. Rather than treating this result as representative of the final model, it motivates the next training stage, in which safety and mixed obligation formulae are incorporated into the task distribution. Taken together, these preliminary experiments indicate that the SemLTL$_f^+$ modifications preserve strong performance on the guarantee fragment on which the model is currently trained, while also showing some zero-shot transfer to unseen logical structures. In particular, the successful transfer to pure safety is encouraging; however, the mixed safety–guarantee result also demonstrates that while safety and guarantee indeed are dual properties, their arbitrary boolean combinations (LTL$_f^+$) gives rise to a more complicated setting. Despite the fact that SemLTL is trained on more complex formula types such as recurrence and persistence, SemLTL$_f^+$ obtains higher mean performance on two of the three structurally unseen categories. *It is noteworthy that since safety and guarantee are dual to each other, it raises the possibility that semantic features learned from guarantee-oriented tasks may also encode information useful for safety-oriented behaviour; establishing such transfer systematically requires a larger controlled evaluation.*

	Task Specification Type	Finite	SemLTL		SemLTL$_f^+$ G	
			SR / μ_{acc}	Length	SR / μ_{acc}	Length
Seen	Existential Reach	✓	0.99 ± 0.01	112 ± 13	0.99 ± 0.01	125 ± 22
	Sequential Reach	✓	0.98 ± 0.02	224 ± 33	0.98 ± 0.00	217 ± 36
	Reach-Avoid	✓	0.96 ± 0.01	110 ± 24	$\mathbf{0.97 \pm 0.01}$	113 ± 22
	Sequential Reach-Avoid	✓	$\mathbf{0.95 \pm 0.04}$	233 ± 34	0.92 ± 0.00	225 ± 31
	General Reach-Avoid	✓	0.87 ± 0.07	203 ± 32	$\mathbf{0.92 \pm 0.00}$	$\mathbf{171 \pm 42}$
Unseen	Conjunctive Guarantee	✓	0.98 ± 0.07	X	$\mathbf{0.99 \pm 0.02}$	X
	Pure Safety	×	0.99 ± 0.01	X	$\mathbf{1.00 \pm 0.01}$	X
	Safety + Guarantee	×	$\mathbf{0.61 \pm 0.01}$	X	0.40 ± 0.05	X

Table 6.2: Evaluation results of SemLTL$_f^+$ (guarantee version) and the baselines. For each task family, SR denotes success rate and length is the average length of an episode until a goal is reached (only for guarantee formulae), while for infinite horizon tasks, it represents the number of accepting runs as a fraction of the total episode length. SemLTL$_f^+$ is trained on "seen" task types while unseen ones are unencountered thus far.

6.3.2 SemLTL$_f^+$ Performance

The standard SemLTL baseline remained slightly stronger on all finite-horizon categories. These results indicate that SemLTL$_f^+$ learned the principal navigation, avoidance, and multi-goal behaviours, but did not quite match the baseline's reliability. The largest difference occurred for sequential reach-avoid specifications. Inspection of the individual formulas showed that the obligation model performed particularly poorly on one of the two nested reach-avoid instances. This suggests that the remaining limitation is not simply a lack of exposure to the formula family. Instead, the deterministic automaton embedding may provide a less effective representation of which subgoal is currently active and which avoidance constraint applies during each part of the sequence. This motivates future work where main and breakpoint formulae can be embedded as an ablation study. Considering the half-size light-weight embedding, it is relatively still competitive in most task types with the baseline.

The matched-mixture curriculum nevertheless improved general reach-avoid performance considerably relative to the earlier obligation model. Explicitly sampling one or two goal propositions and zero to two avoidance propositions increased performance on this category from approximately 0.73 to 0.83 when compared with a previous model that had this task family as unseen. The remaining difference

[3]The semantic label embedding layer is half the size. However, there is a 12% reduction in overall model size.

	Task Specification Type	Finite	SemLTL	SemLTL$_f^+$
			SR / μ_{acc}	SR / μ_{acc}
Seen	Existential Reach	✓	$\mathbf{0.99 \pm 0.01}$	0.97 ± 0.03
	Sequential Reach	✓	$\mathbf{0.98 \pm 0.02}$	0.94 ± 0.03
	Reach-Avoid	✓	$\mathbf{0.96 \pm 0.01}$	0.93 ± 0.03
	Sequential Reach-Avoid	✓	$\mathbf{0.95 \pm 0.03}$	0.84 ± 0.04
	General Reach-Avoid	✓	$\mathbf{0.87 \pm 0.08}$	0.83 ± 0.09
	Conjunctive Guarantee	✓	$\mathbf{0.98 \pm 0.01}$	0.95 ± 0.03
	Pure Safety	×	0.99 ± 0.00	$\mathbf{1.00 \pm 0.02}$
	Safety + Guarantee	×	0.61 ± 0.10	$\mathbf{0.64 \pm 0.11}$

Table 6.3: Evaluation results for SemLTL and SemLTL$_f^+$. For finite-horizon guarantee tasks, SR denotes the proportion of successful episodes. For infinite-horizon tasks, μ_{acc} denotes the mean fraction of an episode spent traversing accepting automaton transitions, computed per evaluation run as accumulated accepting reward divided by episode length. Results report the mean and standard deviation across five random seeds. All task families are represented structurally in the matched-mixture training curriculum.

from SemLTL, which achieved 0.87, was therefore reduced to around four percentage points. This supports the conclusion that a substantial part of the original difference was caused by training-distribution mismatch rather than an inherent limitation of deterministic automata.

For pure safety specifications, both systems spent almost every timestep on accepting transitions. SemLTL$_f^+$ obtained an accepting occupancy extremely close to one and typically survived almost the entire evaluation horizon. Its numerical advantage over SemLTL is small in accepting-occupancy terms, although the obligation model's longer episodes indicate that it encountered forbidden propositions less frequently. **For combined safety-and-guarantee specifications, SemLTL$_f^+$ obtained a higher accepting occupancy than SemLTL**, achieving 0.64 ± 0.11 compared with 0.61 ± 0.10. This suggests that the matched-mixture curriculum improved the model's ability to balance progress towards guarantee objectives while continuing to satisfy safety restrictions. In particular, performance in this category is desirable as this is the primary research objective of this thesis and the most general version of obligations.

Overall, the matched training distribution narrowed the finite-task performance gap but did not remove it. Across the comparable finite categories in the table, SemLTL$_f^+$ achieved an average success rate of approximately 0.91, compared with 0.95 for SemLTL. The remaining difference is concentrated in nested sequential tasks, sug-

gesting that future improvements should focus on state embeddings that more explicitly represent outstanding goals, active avoidance constraints, and progress through the deterministic automaton. **Although the improvement is limited, this performance is achieved by a lighter-weight, faster-trained model that is exposed to less variety of task types.**

Taken together, the results provide encouraging evidence for the proposed framework. The principal observation is not that SemLTL$_f^+$ uniformly outperforms SemLTL, but that replacing the LDBA-based interface with a deterministic weak-automaton representation does not appear to introduce a systematic loss in performance on the trained task distribution. Across the seen task families, SemLTL$_f^+$ remains broadly competitive with SemLTL, and in several cases achieves either a higher success rate or shorter successful trajectories. This is particularly notable given that the current SemLTL$_f^+$ model uses a substantially smaller semantic representation: 167 dimensions compared with 386 for SemLTL. The unseen tasks provide an additional indication of the potential of the semantic representation. In particular, the 1.00 success rate on Pure Safety is noteworthy because the model has not been trained on pure safety specifications at this stage. This should not be interpreted as establishing general zero-shot safety reasoning; nevertheless, it suggests that the logic-derived DWA representation captures information that can support useful behaviour outside the temporal structures explicitly represented during training. Similarly, the strong performance on the unseen Conjunctive Guarantee family indicates that the learned policy is not restricted to reproducing the precise specification structures encountered during training.

The weaker result on the unseen Safety + Guarantee family is also informative. Unlike pure safety, this family requires avoidance behaviour to be coordinated with goal-directed progress within the same specification. Its lower success rate therefore identifies a current boundary of zero-shot transfer for the guarantee-trained model and motivates explicitly incorporating safety and mixed obligations into the training distribution. More broadly, these experiments should be understood as an initial feasibility study for SemLTL$_f^+$ rather than a definitive benchmark for LTL$_f^+$ reinforcement learning. In the absence of an established directly comparable LTL$_f^+$ multi-task baseline, the SemLTL comparison provides evidence that the semantic-automata paradigm can be transferred from LDBAs to deterministic weak automata while retaining effective multi-task learning. The fact that this early implementation is already competitive on its trained task families, while using a considerably smaller

embedding and exhibiting transfer to unseen temporal structures, suggests substantial scope for further improvement through broader training and a capacity-matched semantic representation.

Chapter 7

Conclusion

Temporal specifications provide a powerful interface for reinforcement learning agents whose objectives extend beyond instantaneous rewards. However, using these specifications in multi-task reinforcement learning introduces two distinct representational requirements. First, the agent must retain sufficient memory to determine its progress through a non-Markovian objective. Second, this progress must be represented in a form that permits knowledge to transfer between different temporal tasks. An automaton state addresses the first requirement, but its arbitrary identifier does not, by itself, address the second. This thesis investigated this problem through the perspective of semantically labelled automata. Building on SemLTL, we introduced SemLTL$_f^+$, an adaptation of the semantic-automata paradigm to LTL$_f^+$ obligations. The central premise of SemLTL is preserved: the representation responsible for correctly *monitoring* a temporal objective need not be identical to the representation exposed to the learning algorithm. Instead, the automaton maintains exact temporal progress, while logic-derived features communicate the semantic meaning of that progress to a universal policy.

The principal contribution of this work is to demonstrate how this separation can be realised using a substantially different automata-theoretic backend. Whereas SemLTL translates LTL specifications into semantically labelled Limit-Deterministic Büchi Automata, SemLTL$_f^+$ uses the structure of LTL$_f^+$ obligations to construct semantically labelled Deterministic Weak Automata. This change is particularly natural for the obligation fragment, in which safety and guarantee requirements can be constructed from finite-trace components while still specifying infinite-trace behaviour. The change from LDBAs to DWAs has consequences throughout the architecture. Automaton progression becomes deterministic: for a given automaton state and propositional observation, the next task state is uniquely determined. The agent therefore no longer needs to resolve the restricted automaton nondeterminism

introduced through LDBA ϵ-transitions. In the revised implementation, the corresponding ϵ-action policy component can consequently be removed entirely, leaving the reinforcement learning policy responsible only for actions in the physical environment. The semantic representation was similarly adapted to the structure of the new monitor. SemLTL represents an LDBA state using semantic information associated with main and breakpoint formulae. SemLTL$_f^+$ instead extracts features from a single semantic label associated with each DWA state. The resulting representation retains logic-derived information such as trueness and propositional attention while reducing the semantic feature vector from 386 to 167 dimensions. In the implementation evaluated in this thesis, this also reduces the complete policy parameter count relative to the corresponding SemLTL architecture.

The experimental results provide encouraging evidence that this simpler interface remains capable of supporting temporal multi-task learning. Across the guarantee task families used during training, the preliminary SemLTL$_f^+$ model achieved performance broadly comparable with SemLTL despite operating with a substantially smaller semantic representation. Particularly interesting behaviour was observed under zero-shot evaluation. A model trained only on guarantee-oriented specifications successfully transferred to a pure safety task, despite pure safety formulae being structurally absent from its training distribution. This suggests that the logic-derived embedding can expose behavioural relationships between temporal objectives that extend beyond memorisation of the task families encountered during training.

The same preliminary model was less successful when safety and guarantee requirements were composed within a single unseen specification. This distinction is informative. Pure safety transfer indicates that useful avoidance behaviour can emerge from the shared semantic representation, but successful execution of mixed obligations requires the policy to coordinate avoidance with goal-directed progress. The subsequent expansion of the training curriculum to include pure safety, conjunctive guarantees, general reach–avoid tasks, and mixed safety–guarantee specifications therefore provides a more appropriate test of the complete obligation framework.

Taken together, these findings support the broader hypothesis motivating this thesis: the usefulness of an automaton for reinforcement learning is not determined solely by whether it correctly recognises the temporal language. Its internal structure also determines the interface between symbolic task reasoning and learned control. Deterministic weak automata provide an attractive interface for LTL$_f^+$ obligations because they separate deterministic temporal monitoring from environment control,

while semantic labelling prevents this determinism from reducing the task representation to opaque categorical states.

Several extensions follow directly from these limitations.

Matched-capacity semantic embeddings. A first priority is to control for representation size. The current 167-dimensional DWA embedding could be projected or augmented to match the 386-dimensional representation supplied to SemLTL. Training otherwise identical policies with matched semantic-input dimensions would provide a cleaner comparison between the LDBA and DWA representations. More generally, an ablation over embedding dimensionality could determine whether the compact representation acts as a useful inductive bias or instead creates a capacity bottleneck on more compositionally complex tasks.

Main and breakpoint-style DWA representations. A complementary experiment would retain the DWA monitor while reconstructing a richer semantic interface analogous to SemLTL's main and breakpoint formulae. Although breakpoint formulae are not required for DWA acceptance in the same way that they are required by SemLTL's LDBA construction, a second semantic channel could explicitly represent a locally active sub-obligation or short-term progress target. This would separate two questions that are currently changed simultaneously: whether DWAs provide a useful task monitor, and whether a single semantic formula is sufficient as their policy-facing representation. This gives rise to a particularly informative ablation: LDBA + main/breakpoint vs. DWA + single label vs. DWA + main/breakpoint-style labels. Such a comparison would help identify whether the main/breakpoint decomposition contains transferable information that is useful to the policy independently of the Büchi construction for which it was originally introduced.

Semantic-feature ablations. The contribution of individual semantic features should also be studied systematically. Trueness and propositional attention encode different aspects of task progress: the former characterises the immediate effect of observations, while the latter captures dependencies between propositions that may become relevant later. Removing each feature family independently, and comparing them against raw DWA-state encodings and syntactic formula representations, would quantify how much each contributes to zero-shot generalisation.

Automaton and representation efficiency. The deterministic construction also motivates a more detailed computational comparison. Future experiments could measure automaton construction time, numbers of generated and reachable states, numbers of transitions, memory usage, semantic-embedding computation time, and inference latency. Such measurements would complement success rate by determining whether the structural simplifications of SemLTL$_f^+$ translate into practical computational advantages.

Generalisation to more complex obligations. The preliminary zero-shot safety result suggests that the learned policy can transfer between logically related specification families. A larger evaluation should test this systematically by withholding particular Boolean structures, temporal depths, proposition combinations, and entire obligation families during training. Of particular interest is compositional generalisation: whether behaviours learned separately for safety and guarantee objectives can be combined at test time to solve novel safety–guarantee specifications.

Beyond the obligation fragment. Finally, the present framework could be extended towards richer LTL$_f^+$ specifications, including recurrence and persistence properties outside the safety–guarantee obligation fragment considered here. Doing so would require reconsidering both the automaton construction and the semantic representation, but would provide a natural test of the broader thesis advanced by this work: that semantic automaton states can serve as a general interface between symbolic temporal reasoning and multi-task reinforcement learning.

SemLTL$_f^+$ should therefore be viewed not merely as a replacement of one automaton class with another, but as evidence that the semantic automata paradigm is portable across temporal logics and automata constructions. The original SemLTL framework demonstrated that exposing the logical meaning of LDBA states can support universal policies over LTL instructions. This thesis shows that the same principle can be reformulated around deterministic weak automata for LTL$_f^+$ obligations, yielding a smaller semantic interface and a cleaner separation between temporal monitoring and physical control. The current results leave room for further optimisation, particularly in the design and capacity of the semantic embedding. Nevertheless, the fact that a substantially reduced representation remains competitive on trained task families and exhibits transfer to temporal structures absent from its training distribution is a promising result. Rather than requiring a particular automaton formalism, semantic

task conditioning appears to offer a broader design principle: use symbolic machinery to maintain correct task progress, but expose the *meaning* of that progress instead of an arbitrary symbolic state to the learning agent.

The results of this thesis can be understood directly in relation to the research questions posed in Chapter 1. First, with respect to **RQ1**, this work demonstrates that the semantic-labelling paradigm can be successfully reformulated for LTL_f^+ obligations using Deterministic Weak Automata. The resulting SemLTL_f^+ framework preserves the separation between symbolic task monitoring and policy-facing semantic representation while adapting the automaton construction, state labelling, and reinforcement-learning interface to the deterministic obligation setting. Second, with respect to **RQ2**, the experiments show that this representation is capable of supporting multi-task learning and generalisation across temporal specifications. SemLTL_f^+ remains broadly competitive with SemLTL on the task families represented during training, while the guarantee-oriented preliminary model also exhibits transfer to structurally unseen specification types, including conjunctive guarantees and pure safety objectives. The final matched-mixture model further demonstrates that safety and combined safety–guarantee obligations can be learned within the same framework, although performance on several finite-horizon and nested sequential tasks remains below the SemLTL baseline. Finally, **RQ3** concerns the trade-offs introduced by replacing the general-LTL LDBA pipeline with the specialised DWA construction. The principal gain is a cleaner separation between temporal monitoring and physical control: automaton progression is deterministic, so the policy no longer needs to resolve ϵ-transition choices, and the semantic interface is reduced from 386 to 167 dimensions while retaining competitive performance across much of the evaluated task distribution. The corresponding cost is that the simpler single-label representation appears less effective for some compositionally structured and nested tasks, suggesting that information captured by SemLTL's richer main/breakpoint representation may remain useful to the policy. The present experiments therefore answer the central question of this thesis positively but with an important qualification: semantically labelled DWAs provide a viable and promising task representation for multi-task reinforcement learning with LTL_f^+ obligations, but the current results do not yet establish a computational advantage in wall-clock efficiency, nor do they isolate whether the remaining performance gap is caused by the DWA monitor itself or by the reduced capacity of its semantic representation.

Limitations

The results should nevertheless be interpreted in light of several limitations. Most importantly, SemLTL$_f^+$ and SemLTL do not currently receive equal-capacity semantic representations. The proposed model uses a 167-dimensional feature vector, compared with 386 dimensions for SemLTL. Consequently, the comparison simultaneously changes the automaton construction and reduces the information capacity of the policy-facing task representation. Competitive performance under this constraint is encouraging, but it prevents the experiments from isolating whether remaining performance differences originate from the DWA construction itself or simply from differences in representational capacity. The two frameworks also represent semantic state information differently. SemLTL explicitly maintains main and breakpoint formulae, whereas the current SemLTL$_f^+$ implementation embeds a single formula label associated with each DWA state. The experiments therefore evaluate a deliberately simpler semantic interface rather than a direct feature-for-feature translation of SemLTL onto DWAs.

Finally, the empirical evaluation covers only a subset of the temporal structures expressible by the broader LTL$_f^+$ formalism. The present work focuses on the obligation fragment and principally on safety, guarantee, and their compositions. The results should therefore be viewed as evidence for the suitability of semantically labelled DWAs within this setting rather than as an empirical claim covering arbitrary LTL$_f^+$ specifications.

References

[1] Alessandro Abate, Giuseppe De Giacomo, Mathias Jackermeier, Jan Kretínský, Maximilian Prokop, and Christoph Weinhuber. Semantically labelled automata for multi-task reinforcement learning with ltl instructions, 2026.

[2] Oguzhan Alagoz, Heather Hsu, Andrew J Schaefer, and Mark S Roberts. Markov decision processes: a tool for sequential decision making under uncertainty. *Medical Decision Making*, 30(4):474–483, 2010.

[3] Benjamin Aminof, Giuseppe De Giacomo, Sasha Rubin, and Moshe Y Vardi. Ltlf+ and ppltl+: extending ltlf and ppltl to infinite traces. *arXiv preprint arXiv:2411.09366*, 2024.

[4] Fahiem Bacchus, Craig Boutilier, and Adam Grove. Rewarding behaviors. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1160–1167, 1996.

[5] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22(1):5–27, 1998.

[6] Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

[7] Alper Kamil Bozkurt, Yu Wang, Michael M Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10349–10355. IEEE, 2020.

[8] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[9] Antonio Casares, Thomas Colcombet, Nathanaël Fijalkow, and Karoliina Lehtinen. From muller to parity and rabin automata: Optimal transformations preserving (history) determinism. *TheoretiCS*, 3, 2024.

[10] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, Sasha Rubin, et al. Pure-past linear temporal and dynamic logic on finite traces. In *IJCAI*, pages 4959–4965, 2020.

[11] 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 international conference on automated planning and scheduling*, volume 29, pages 128–136, 2019.

[12] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. 2013.

[13] Javier Esparza, Jan Křetínskỳ, Jean-François Raskin, and Salomon Sickert. From ltl and limit-deterministic büchi automata to deterministic parity automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 426–442. Springer, 2017.

[14] Javier Esparza, Jan Křetínský, and Salomon Sickert. A unified translation of linear temporal logic to w-automata. *J. ACM*, 67(6), October 2020.

[15] Javier Esparza and Jan Křetínský. From ltl to deterministic automata: A safraless compositional approach, 2014.

[16] Giuseppe De Giacomo, Christian Hagemeier, Daniel Hausmann, and Nir Piterman. Symbolic synthesis for ltlf+ obligations, 2026.

[17] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

[18] 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 *2019 IEEE 58th conference on decision and control (CDC)*, pages 5338–5343. IEEE, 2019.

[19] Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Deep reinforcement learning with temporal logics. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–22. Springer, 2020.

[20] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

[21] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

[22] Mathias Jackermeier and Alessandro Abate. Deepltl: Learning to efficiently satisfy complex ltl specifications for multi-task rl. In *International Conference on Learning Representations*, volume 2025, pages 14000–14028, 2025.

[23] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[24] Jan Křetínskỳ, Alexander Manta, and Tobias Meggendorfer. Semantic labelling and learning for parity game solving in ltl synthesis. In *International Symposium on Automated Technology for Verification and Analysis*, pages 404–422. Springer, 2019.

[25] Ariel Kwiatkowski, Mark Towers, JK Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Kallinteris Andreas, Markus Krimmel, Arjun KG, et al. Gymnasium: a standard interface for reinforcement learning environments. 2024.

[26] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y Vardi, and Jifeng He. Ltl satisfiability checking revisited. In *2013 20th International Symposium on Temporal Representation and Reasoning*, pages 91–98. IEEE, 2013.

[27] Zohar Manna and Amir Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):97–130, 1991.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[29] Ryohei Oura, Ami Sakakibara, and Toshimitsu Ushio. Reinforcement learning of control policy for linear temporal logic specifications using limit-deterministic generalized büchi automata. *IEEE Control Systems Letters*, 4(3):761–766, 2020.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[31] Amir Pnueli. The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)*, pages 46–57. ieee, 1977.

[32] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[33] Michael O Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

[34] Dorsa Sadigh, Eric S Kim, Samuel Coogan, S Shankar Sastry, and Sanjit A Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 1091–1096. IEEE, 2014.

[35] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[37] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínskỳ. Limit-deterministic büchi automata for linear temporal logic. In *International Conference on Computer Aided Verification*, pages 312–332. Springer, 2016.

[38] Fabio Somenzi and Roderick Bloem. Efficient büchi automata from ltl formulae. In *International Conference on Computer Aided Verification*, pages 248–263. Springer, 2000.

[39] Richard S Sutton, Andrew G Barto, and Andrew Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[40] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[41] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[42] Pashootan Vaezipoor, Andrew C Li, Rodrigo A Toro Icarte, and Sheila A Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, pages 10497–10508. PMLR, 2021.

[43] Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *1st Symposium in Logic in Computer Science (LICS)*. IEEE Computer Society, 1986.

[44] Nelson Vithayathil Varghese and Qusay H Mahmoud. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9):1363, 2020.

[45] Andrew Wang, Andrew C Li, Toryn Q Klassen, Rodrigo Toro Icarte, and Sheila A McIlraith. Learning belief representations for partially observable deep rl. In *International Conference on Machine Learning*, pages 35970–35988. PMLR, 2023.

[46] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[47] Christoph Weinhuber, Maximilian Prokop, Giuseppe De Giacomo, and Moshe Y. Vardi. Infinite trace objectives with finite trace techniques: Translating ltl to ltlf+, 2026.

[48] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022, 2007.

[49] Lim Zun Yuan, Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Modular deep reinforcement learning with temporal logic specifications, 2019.