Utrecht University

MASTER'S THESIS

Explainable Online Reinforcement Learning Using Abstract Argumentation

Author:
Cándido Otero Moreira
(Nº 6786170)

Supervisors:
Dennis Craandijk
Floris Bex

A thesis submitted in fulfillment of the requirements for the degree of MSc in Artificial Intelligence

in the

Faculty of Science

September 28, 2022

UTRECHT UNIVERSITY

Abstract

Faculty of Science

MSc in Artificial Intelligence

Explainable Online Reinforcement Learning Using Abstract Argumentation

by Cándido Otero Moreira

The democratisation of deep learning (DL) in recent years has led to an increasing presence of DL algorithms influencing our everyday lives, from recommending us our next book to deciding whether we are granted a loan or not. Although DL has allowed for a major performance boost in data-driven applications, the decisions made by neural networks are completely opaque to humans, rendering their suitability questionable for applications where the model needs to be verifiable and/or explanations must be completely faithful to the model. Related literature exists that tries to overcome this problem by using model extraction to derive an (approximately) equivalent symbolic model using a value-based argumentation framework (VAF) as its inference engine. While the resulting model has the advantage of being verifiable and providing faithful explanations, model extraction imposes an exploration boundary on the symbolic model. This thesis proposes a novel approach that integrates formal argumentation in an end-to-end reinforcement learning (RL) pipeline. The benefit of this method is that the model can be trained using online RL instead of using a surrogate model, leading to a potentially better solution while still using a VAF as its inference engine.

Acknowledgements

I want to thank Dennis and Floris for their supervision throughout this project and for helping me see some of the challenges it entailed in a different light. I am also thankful to the Politielab team for letting me share my ideas with them and offering me their feedback. Finally, I am grateful to my family and friends for supporting me during these unpredictable last years.

Contents

List of Abbreviations

AA	**A**bstract **A**rgumentation
AI	**A**rtificial **I**ntelligence
AARL	**A**rgumenation **A**ccelerated **R**einforcement **L**earning
AF	**A**rgumentation **F**ramework
BAF	**B**ipolar **A**rgumentation **F**ramework
CO	**C**ombinatorial **O**ptimisation
DL	**D**eep **L**earning
FFL	**F**oggy **F**rozen **L**ake
FL	**F**rozen **L**ake
GDPR	**G**eneral **D**ata **P**rotection **R**egulation
GOFAI	**G**ood **O**ld-**F**ashioned **A**rtificial **I**ntelligence
HA	**H**andcrafted **A**gent
KB	**K**nowledge **B**ase
MDP	**M**arkov **D**ecision **P**rocess
ML	**M**achine **L**earning
NSA	**N**on-**S**ymbolic **A**gent
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
RA	**R**andom **A**gent
RL	**R**einforcement **L**earning
RNN	**R**ecurrent **N**eural **N**etwork
SA	**S**ymbolic **A**gent
SAF	**S**ituation-specific **A**rgumentation **F**ramework
SRA	**S**urroundings-aware **R**andom **A**gent
VAF	**V**alue-based **A**rgumentation **F**ramework
VSAF	**V**alue-based **S**ituation-specific **A**rgumentation
xAI	E**x**plainable **A**rtificial **I**ntelligence

1 Introduction

The last decade has seen some major breakthroughs in AI such as surpassing human-level performance in image tagging [19] or beating the human champion of the board game Go [50]. These achievements are in part thanks to the availability of massive amounts of data to train supervised classifiers and to improvements in hardware, which have allowed for a dramatic reduction of training time by running deep learning (DL) algorithms on powerful GPUs [38].

Despite these advances, a common drawback of most current connectionist models (neural networks) is still their opacity in decision-making tasks. This hinders the deployment of such models in applications where safety is paramount; for instance, terrorism detection [47] and medical diagnostic systems [14]. Additionally, the so-called "right to an explanation" derived from the General Data Protection Regulation (GDPR) enforced in Europe since 2018, requires that any automated decision based on an individual's personal information can be supported by the corresponding argument(s) that brought about that decision [26]. These are some of the reasons why researchers like Gerlings et al. [25] regard explainable artificial intelligence (xAI) as a pressing matter in the AI research agenda.

It is almost impossible to talk about explainability without talking about interpretability. Both concepts are closely related and some scholars even use them interchangeably. A good account of the connection between the two is given by Biran and Cotton [8]: "*Explanation is closely related to the concept of interpretability: systems are interpretable if their operations can be understood by a human, either through introspection or through a produced explanation*". Indeed, one trend observed in the literature to improve interpretability is through the introspection of neural networks by adding visualisations or by translating them into symbolic models [60]. This allows us to trace back and examine the steps followed by a connectionist model when it makes a decision. The other trend to improve interpretability is through self-explainable AI [18], where the model is designed to produce explanations that support its decisions (e.g., a medical diagnostic system may output its prediction along with a set of low-level labels that support the diagnosis and that can be understood by the medical practitioner [49]). The interpretability of these models will increase as the quality of their explanations improves; therefore, it is natural to wonder what constitutes a *good* explanation. Jacovi and Goldberg [29] argue that a *good* explanation is not the most plausible (the aim of the explanation is not to convince the human), but the most faithful (the one that most accurately captures the reasoning process of the model). However, as Feng et al. [20] point out, high-fidelity explanations can result in nonsensical arguments to the user. Philosophers and social scientists have also discussed what constitutes a *good* explanation [10] [27] [41]. Although the definition of the *golden* explanation remains up for debate, Oestermeier and Hesse [44] showed that the most common explanations use *mechanistic causal arguments* [1] of the form: "A caused C because A led to C via mechanism B". In their study, they examined different German corpora screening for causal claims, and mechanistic causal evidence accounted for 75.1% of all the claims, whereas the next most common class of arguments had an incidence of only 4.3%.

A popular method to improve interpretability is to ascribe the model to some expert knowledge. This knowledge can be represented by crafting a knowledge base (KB), which, simply put, is a collection of rules (if-then-else) given by a domain expert. KB systems (or expert systems) gained popularity in the 80s and are central to symbolic AI (also known as *good old-fashioned AI* or GOFAI). Connectionist models and GOFAI are at opposite ends of the spectrum: while connectionist models are flexible universal function approximators, GOFAI is concerned with using a set of fixed rules as its inference mechanism [21]. Several instances of connectionist models combined with a KB exist in the recent literature, from fake news detectors [28] to medical diagnostic systems [49]. Tiddi and Schlobach [56] present a survey of models that leverage expert knowledge to improve the interpretability in machine learning (ML). Expert knowledge not only helps to make the model more interpretable, but it can also improve performance. There are two main mechanisms through which knowledge injection can improve model performance. The first one is *feature extraction*, which consists in using the domain knowledge to engineer new features by reasoning about the original ones. This can boost accuracy in classifiers when data is scarce and/or when the function to be learned is very complex. Borghesi et al. [9] report an average accuracy improvement of 38% in their model after injecting it with domain knowledge, with respect to their purely data-driven model. The other main mechanism to improve model performance found in the literature is to guide exploration in the reinforcement learning (RL) paradigm. In online RL, an agent explores the environment intending to maximise a numerical signal (reward) emitted by the environment. When the rewards are very sparse (infrequent), the agent usually takes a long time to learn a good policy that allows it to maximise the accumulated reward over time. One way to alleviate this problem is by recommending the agent to explore some (suboptimal) set of state-action pairs according to the rules provided by the KB. One such application of KB in RL is found in the work of Gao et al. [22], in which it yielded faster convergence and a higher accumulated reward on average. Apart from improvements in accumulated reward and convergence time, literature exists that reports using domain knowledge to improve the generalisation capabilities of their models in diverse data-driven applications, ranging from text classification [35] to transformer protection in power grids [37]. *Domain generalisation* [61] is important when a model is presented in production with problems that come from a distribution different from the instances it saw during training.

Since expert knowledge is given in the form of a collection of rules, some ML researchers [16] view these rules as arguments that guide the model to improve its performance, training time or interpretability. As a conceptual example, imagine that an agent is learning to drive a car using RL. One possible argument A for this task could be: *"if the traffic light is green, then move forward"*. This means that if at time t the state s_t of the environment is such that the premise "the traffic light is green" is true, argument A will conclude that the best possible action a_t for the agent to execute is to move forward. Now consider this other argument B: *"if there is a pedestrian ahead, then stop"*. It is clear that both arguments (A and B) can be simultaneously valid (i.e., a reckless pedestrian may be crossing the road while the traffic light is green for the car), but they promote contradictory actions. Since arguments can be in disagreement, some conflict resolution mechanism is needed to identify a unique winning argument at any given state of the environment. Dung's seminal work [17] laid the foundations of abstract argumentation frameworks (AFs), which are a popular formalism to handle conflicting arguments in abstract argumentation

(AA). AA tries to capture the way humans reason in presence of incomplete information and/or when a new piece of information contradicts our previous knowledge. According to Wyner et al. [59], AFs can be used to instantiate a KB. The basic idea behind AFs is that a set of arguments can be represented as nodes in a graph and the attacks between arguments can be represented as directed arrows (from the attacker to the attacked argument). By using some criteria (argumentation semantics), it can be determined which set of arguments can be accepted. Dung's original formulation of AFs assumes that all attacks have the same strength, which might not be enough to resolve all scenarios (i.e., conflict resolution can end in a tie). To overcome this problem, variants of the original definition of AFs conceived by Dung have been proposed, such as value-based argumentation frameworks (VAFs) [7], where the defeasibility of arguments depends on their associated value. Liao et al. [36] defend this type of agent from the point of view of ethics: using a value-driven argumentation-based agent to represent its reasoning process is enough to justify and explain its actions.

This section has made a case for the relevance of xAI in modern research by summarising the concerns of some AI scholars [25]. A definition of *explainability* has been given [8], citing a few works that improve *interpretability* either by using symbolic models [60] or by accompanying their model prediction with an explanation [49]. One such way of improving *introspection* (understood as per the mentioned definition given by Biran and Cotton [8]) and performance is through the use of a KB, and several studies using hybrid (connectionist and symbolic) models were mentioned [28] [49] [9] [56]. Finally, AFs [17] were introduced and proposed as a plausible way of instantiating a KB, as supported by the literature [22] [32]. The main goal of this thesis is to create an end-to-end RL pipeline that trains a value-driven agent that uses an AF to instantiate domain expert knowledge. The outcome is a symbolic agent that uses the learnt VAF as its inference engine. In the remainder of this chapter, the strengths and weaknesses of the main related works are outlined and compared to our approach. Finally, the research questions of this study are presented and an overview of the remaining chapters of this dissertation is given.

1.1 Background

A comprehensive review of existing literature using KBs to build explainable AI models can be found in the work of Tiddi and Schlobach [56]. Cocarascu and Toni [16] also extensively examine relevant studies that use formal argumentation to improve performance and/or interpretability of ML models. Three specific studies [57] [22] [32] that combine RL with some KB motivate this thesis. This section briefly explains each of these 3 works and highlights the advantages and disadvantages of each of them (the complete discussion of the related literature will be given in Chapter 3). This will serve as a basis to support a novel KB-RL approach that will be proposed in the next section.

As mentioned in the previous section, the use of some form of KB can aid RL by improving its accuracy, robustness, convergence time and/or by making the model (more) interpretable [56] [16]. One example of such a performance boost can be achieved through argumentation accelerated reinforcement learning (AARL), as reported by Gao et al. [22]. In their work, AARL is used in a multi-agent setting to learn to play a subtask of RoboCup, a soccer simulator. As the name suggests, the goal of AARL is to speed up the learning process. This is achieved by exploring the environment according to some heuristics determined by a VAF created by the

domain expert. For example, coming back to the self-driving car example from the previous section, argument *B* will increase the chances that the agent will explore the action "stop the car" if there is a pedestrian in front of the vehicle because that action is likely to return a high reward. However, there might be better actions to be taken in that scenario. It may seem obvious that a car should stop if a pedestrian is in front of it, but imagine that other sensors are also measuring the position and velocity of both the car and the pedestrian. It could be the case that the car is still far from the pedestrian and, given the velocity of both subjects, it is actually safer for the car to slowly decelerate than to brake sharply (perhaps even causing the collision of other cars behind it). That is why AARL can provide both higher rewards and faster training, because it leverages the KB while leaving room for exploring better actions. Although AARL is an excellent example of KB injection to improve performance in RL, its benefits in terms of interpretability are very limited. The injected VAF provides a template to explore the environment (which may give some insight into how the agent behaves), but, since the inference engine of the agent is a universal function approximator (a neural network), the resulting policy learnt by the agent can be very different from that dictated by the VAF.

One way of alleviating this lack of transparency could be to gather a vast collection of expert rules and have the RL agent learn which one it should execute at any given state. This is precisely what Voss et al. [57] did to train an RL agent to play FreeCiv (an open-source empire-building strategy game based on the popular Civilization). They gathered thousands of rules from several different human players and created a KB (no AF was used in their method for KB instantiation). Each rule of their KB has a set of conditions (premises) and some procedural knowledge (the conclusion), such that, when the conditions are met (according to the state of the game and the agent's memory), the rule becomes active and its procedural knowledge dictates what action(s) should be executed by the agent (this approach will be referred to as KB-inference). Having such a large pool of rules coming from different experts, clashes among rules are expected to occur. Hence, some conflict resolution method is needed to decide which rule's procedural knowledge will be applied. In their approach, an agent is trained using RL to choose only one of the conflicting rules based on the raw features of the game (e.g., score, population size, resources, etc). According to Nechepurenk et al. [42], this approach was tested on a subtask of FreeCiv and yielded a reward 13% higher than their purely connectionist model. This approach is more interpretable than a connectionist model because the rules explicitly state the conditions that need to be met to apply some given procedural knowledge. However, when a clash among multiple rules occurs, the conflict is resolved via a neural network, which obfuscates the explanation of why one rule is chosen over the other(s). In other words, the *mechanism* [1] by which the conflict was resolved involves the evaluation of a neural network, whose reasoning is generally opaque to humans.

The interpretability of such KB-inference techniques can be improved by using a more transparent conflict resolution mechanism. This is exactly what Kazhdan et al. [32] achieved with MARLeME, a model extraction library for multi-agent RL. MARLeME is a library that takes two inputs, a set of RL trajectories and an AF, and identifies the corresponding VAF that best fits the input trajectories. Conceptually, MARLeME works in the opposite way to the AARL approach of Gao et al. [22]: while AARL takes a VAF as input and outputs a connectionist agent, MARLeME takes the trajectories of a connectionist agent as inputs and outputs the VAF that best approximates those trajectories. MARLeME is model-agnostic and only needs to be fed the trajectories of the RL agent and the AF created by the domain expert.

The authors of MARLeME claim that the resulting VAF can be used in two different ways: (1) keeping the original RL agent deployed in production and periodically inspecting its learnt policy; or (2) deploying the symbolic model and using the VAF as its inference engine. Each option entails a compromise between performance and interpretability. For example, (1) could be a good approach in applications where absolute performance is key and the faithfulness of the explanation is not crucial to the operation. In such situations, it would be possible to periodically collect a set of trajectories from the agent and learn a VAF with them. This can be useful to study how the policy evolves with time or how different AFs fit the learnt behaviour of the agent. However, (2) would be preferred in applications where safety and/or interpretability are critical since the behaviour of the VAF can be verified.

Of the three approaches reviewed in this section, MARLeME [32] is the winner in terms of interpretability. On the one hand, if an agent is trained using AARL [22], a symbolic model can always be extracted using MARLeME to study the learnt policy (without replacing the original one). Even if the faithfulness of the extracted model is not guaranteed, it allows some degree of introspection of the neural network. The original model can still be used in production, so using MARLeME just as a post-hoc explanation method has no negative impact in performance. On the other hand, compared to the KB-inference approach of Voss et al. [57], using the VAF identified by MARLeME to resolve conflicts among arguments is an improvement in interpretability. This is because value-driven approaches offer a better explanation since the conflicts can now be resolved transparently, according to the value of each argument. In this case, it is not guaranteed that using a VAF to resolve the clashes among rules will not affect performance. This is because the VAF assumes that the ordering of the arguments is fixed, but the RL agent uses the raw features of the game to determine the winner rule at each state, which allows for a more flexible arbitration. Therefore, there is no guarantee that the preference of one rule over others will be optimal for every situation. This is a recurring compromise between performance and interpretability that is present in all these works (also in ours).

1.2 Research Questions

In the previous section, three studies that use some form of knowledge-based RL technique have been presented, being MARLeME [32] the approach that offered the best properties in terms of interpretability. As mentioned in the previous section, the model extracted by MARLeME can be used either to study the policy learnt by the RL agent or to replace the original model altogether. The former has the disadvantage that the extracted model is not guaranteed to provide *faithful explanations* (as defined by Jacovi and Goldberg [29]) of the decisions made by the original model. On a multi-agent RL task, the authors of MARLeME report fidelity scores of 0.86, 0.84 and 0.68 for the 3 models extracted by MARLeME. The fidelity score is a metric (ranging from 0 to 1, being 1 the highest degree of fidelity) that they used to measure how well an extracted model mimics the original one. The fidelity score can be improved by providing MARLeME with a more expressive AF. The alternative of replacing the original model with the model extracted by MARLeME has the disadvantage of deploying an agent that has been trained on a surrogate agent (via a finite set of generated trajectories), instead of on the real environment. This means that the resulting VAF reflects the ordering that best fits the trajectories of the original agent, not the ordering that necessarily yields the highest reward when interacting with the environment. Another way of looking at it is by defining a policy space Π_{traj} as

the set of all policies that can be derived from the supplied trajectories using topological sorting (the extraction method used by MARLeME) given any possible AF. Because the state-action-reward space can be vast in real applications (or infinite in the continuous domain), the supplied trajectories are usually just a representation of the full space. Hence, the policy space when considering the full state-action-reward space of the original environment, Π_{env}, must be a superset of Π_{traj}. Based on this, a symbolic agent that can explore the original state-action-reward space of the environment can potentially find a better policy than another symbolic agent that is bounded by imitation learning (the extracted model imitates the original model).

In applications where safety and interpretability are paramount [47] [14], researchers may decide beforehand that the deployed agent will use a VAF as its inference engine, to ensure the verifiability of the model. In such cases, one approach would be to translate the arguments of the AF into state features to directly interact with the environment (so it can explore the true state-action-reward space) and learn the ordering of arguments that maximises the accumulated reward (i.e., learn the VAF). This way, the learned model is the same as the model that will be deployed, ensuring verifiability and faithful explanations. This is exactly the main idea behind the novel approach proposed in this thesis: given some AF supplied by a domain expert, we want to learn the best VAF through direct exploration of the environment. Our proposed approach can be summarised in four main steps: (1) designing a game we want to solve; (2) defining an AF that contains useful domain knowledge for an agent to solve the game; (3) learning the best VAF by finding the best ordering of the arguments in the AF; and (4) evaluating the VAF as an inference engine for the game. Step (3) requires the direct exploration of the game, as opposed to MARLeME [32], which used model extraction. The novelty of this approach is that, to the best of our knowledge, this is the first time that a VAF is being learnt directly by interacting with the environment (in our case, the *environment* is the *game* mentioned above).

This section has focused on identifying the disadvantages of model extraction and proposed a novel approach to learning a VAF through direct exploration of the environment, which has the advantage of exploring the full state-action-reward space. To synthesise the purpose of this project, the main research question is formalised as follows:

Research Question 1. Given an RL environment and some KB instantiated in the form of an AF: is it possible for the agent to explore the environment and learn a set of argument values such that the associated VAF maximises the accumulated rewards emitted by the environment?

Three secondary questions can be naturally formulated following the outcome of Research Question 1:

Research Question 2. Can the resulting model be introspected to explain its actions?

Research Question 3. How does the resulting symbolic agent compare to a non-symbolic agent in terms of absolute performance?

Research Question 4. Can the resulting symbolic agent generalise to new environments better than non-symbolic RL agents?

1.3 Structure

This thesis starts by covering in Chapter 2 some preliminary knowledge that will be necessary to understand the details of the related literature and the proposed

approach. Specifically, the present work makes use of concepts from abstract argumentation, reinforcement learning and combinatorial optimisation. Chapter 3 delves deeper into the related KB-RL literature that was presented in Section 1.1. Chapter 4 explains each of the four steps central to our approach (game design, AF definition, learning the VAF and evaluation) by progressively introducing each of the blocks that constitute the proposed pipeline. The conducted experiments are summarised in Chapter 5, analysing the results of each of them separately. Finally, Chapter 6 concludes by summarising the main results of this project and outlining the main challenges encountered in this project and potential future lines of research.

2 Preliminaries

This chapter aims to establish a basic ground on two fundamental concepts that are recurrent in the rest of this work, namely abstract argumentation (AA) and reinforcement learning (RL). Chapters 3 and 4 contain explanations of more complex concepts that build on the fundamentals outlined in this chapter.

2.1 Abstract Argumentation

Daily human reasoning often requires being able to draw conclusions based on incomplete information and/or exceptions to our general beliefs. For these reasons, non-monotonic reasoning is often said to be a requirement for strong AI to develop [53].

Important work towards this direction has been done to create systems capable of performing non-monotonic reasoning in presence of incomplete, contradictory information, such as the systems of Pollock [46] in the late '80s. However, it is probably Dung's theory about argumentation frameworks [17] what marked a breakthrough in this field in 1995. The main idea of his contribution is that an argumentation framework can be modelled as arguments represented by nodes in a graph and their attacks can be represented through directed arrows. Given such a graph, it is possible to define different semantics, which allows the study of the acceptability of its arguments. Intuitively, semantics can be regarded as some eligibility criteria to select a coherent set of arguments (an extension). These selected arguments are considered *accepted* under this particular semantics.

2.1.1 Argumentation Frameworks

According to Dung [17], an argumentation framework (AF) is defined as follows:

Definition 1 (Argumentation framework)**.** An argumentation framework (AF) is a pair (Arg, Att) where Arg is a set of arguments and $Att \subseteq Arg \times Arg$ is a binary relation that defines the attacks between arguments. For any two arguments a and b, such that $(a, b) \in Att$, it is said that a attacks b.

This definition of AF was further extended by Bench-Capon [7] to create the concept of a value-based argumentation framework (VAF).

Definition 2 (Value-based argumentation framework)**.** A value-based argumentation framework (VAF) is a 5-tuple $(Arg, Att, V, val, valpref)$ where Arg and Att define a standard argumentation framework AF, V is a non-empty set of values, val is a function which assigns elements from Arg to elements of V and $valpref$ establishes a preference relation (transitive, irreflexive and asymmetric) on $V \times V$. An argument arg is said to promote a value v if $val(arg) = v$. It is the case that for every $arg_i \in Arg, val(arg_i) \in V$.

This extended definition introduces the notion that some arguments have a higher value than others, in such a way that given the attacks (a, b) and (b, a), a

defeats b iff $val(a) > val(b)$. This contributes to the idea that some arguments may not hold in a given situation in the presence of stronger arguments, but they may hold in a different scenario.

2.1.2 Semantics

An argumentation semantics (or simply semantics) defines zero or more sets of acceptable arguments [3]. Each of these sets of coherent arguments is also called an extension. Two main approaches are used to formulate semantics: extension-based methods and labelling-based methods. In this project, only extension-based semantics will be considered.

When defining a semantics it is useful to define some recurrent concepts that express some relevant relations among arguments.

Definition 3 (Conflict-free set)**.** Given an argumentation framework (Arg, Att), a set S is conflict-free if there are no two arguments $a, b \in S$ such that $(a, b) \in Att$.

Definition 4 (Argument defended by a set)**.** Given an argumentation framework (Arg, Att), a set S defends argument a if for each $b \in Arg$, if $(b, a) \in Att$, b is attacked by S.

Definition 5 (Admissible set)**.** A set of arguments S is admissible if it is conflict-free and for each argument $a \in S$ it is the case that S defends a.

Dung's original work [17] defines four different semantics given an argumentation framework (Arg, Att) and an extension $S \subseteq Arg$:

- **Complete extension**: S is a complete extension if it is an admissible set and for every argument a defended by S it is the case that $a \in S$.
- **Preferred extension**: S is a preferred extension if it is a maximal (with respect to the set inclusion) complete extension.
- **Stable extension**: S is a stable extension if it is a complete set and for every $a \in S$, it is the case that $\forall b \notin S : (a, b) \in Att$.
- **Grounded extension**: S is a grounded extension if it is the minimal (with respect to the set inclusion) complete extension.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a popular paradigm in machine learning, along with supervised learning and unsupervised learning. Simply put, RL is learning what actions to perform in a given environment to maximise some accumulated numerical reward signal [54].

In an RL problem, there is at least one learning agent A that interacts with the environment Env by performing actions on it. After each interaction, the environment evolves (either switching to a different state or remaining at the same one) and returns a reward signal that the agent needs to maximise over time.

2.2.1 Modelling the environment

RL problems are often modelled as (finite) Markov decision processes (MDPs). An MDP is an idealisation of an RL problem in which the probabilities that an environment evolves in a particular way and returns a particular reward signal, depend only on the previous state [54]. More formally:

Definition 6 (Markov decision process)**.** A Markov decision process (MDP) is a 3-tuple (S, A, p), where S is the state space, A is the action space and $p(s', r|s, a)$ is a function that determines the probability of transitioning to a state s' producing a reward signal r when action a is taken from state s.

Modelling an RL problem as an MDP is sometimes too optimistic, as an MDP assumes that the true state of the environment is available to the agent. However, in real-world scenarios, it is common that only a partial observation of the state of the world is available to the agent (e.g., the sensors of a robot may only provide information up to a certain range and the accuracy of their measurements is limited). In such cases, it is best to talk about observations (what is observable by the agent) and states (the full state of affairs of the environment). This is formally known as a partially observable Markov decision process (POMDP):

Definition 7 (Partially observable Markov decision process)**.** A partially observable Markov decision process (POMDP) is a 5-tuple (S, A, O, p, Ω), where (S, A, p) is the MDP that defines the internal dynamics of the process, O is the observation space and $\Omega(o|s, a)$ is a function that determines the probability of outputting observation o from state s when performing action a.

2.2.2 Learning from experience

By interacting with the environment for enough episodes, the agent can learn a good policy that maximises the accumulated reward signal. There are two main methods to achieve this:

1. **Value-based methods**: the aim is to learn a function $q_\pi(s, a)$ that outputs a numeric value that estimates the "quality" (hence the q nomenclature), of being in state s and performing action a according to policy π. To this end, the agent iteratively interacts with the environment and estimates the expected return $v_\pi(s)$ at each state s. The higher the value, the more likely it is for the agent to receive a high reward in the future. This is a form of dynamic programming that is based on the Bellman equation [5]. More formally, $q_\pi(s, a)$ can be defined as:
$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.1}$$
where $\gamma \in [0, 1]$ is a discount factor used to give less weight to subsequent rewards.

2. **Policy gradient methods**: instead of learning about the quality of each state, the agent learns the optimal policy directly by updating the parameters of the policy π by iteratively interacting with the environment. These methods have some advantages over value-based methods, such as being able to learn the probabilities of taking each action or being able to naturally handle continuous action spaces.

There exist other criteria to classify RL agents (although not applicable to all classes of agents) [54]. For example:

- **On-policy vs off-policy**: on-policy methods (e.g., SARSA) iteratively update the same policy they use to interact with the environment (behaviour policy). This is not the case in off-policy methods (e.g., Q-learning), where the behaviour policy is different from the optimal policy that is being learnt.
- **Online vs offline**: online methods are those in which the experience is obtained by the agent interacting directly with the environment. In the case of offline methods, a data set of trajectories (lists of sampled (s, a, r) tuples that describe a possible episode) are supplied to the agent.
- **Tabular methods vs function approximation methods**: in value-based methods, $q_\pi(s, a)$ can be implemented by using a tabular approach or function approximation. The tabular approach requires exploring the different state-action pairs combinations and updating their value accordingly. In cases where the state-action space is too large and exploration is unfeasible, a low-dimensional representation of the state is preferred. This representation can be fed into a function approximator (e.g., a neural network) to estimate the value of each possible action. This makes sampling more efficient since it allows for extrapolation to unexplored states.

3 Related Work

This chapter expands upon the literature overview from Section 1.1 and builds on the basics of AA and RL given in Chapter 2. First, the potential benefits of KB injection in ML are presented, explaining how Voss et al. [57] use a KB-RL setup to teach an agent to play the strategy game FreeCiv. Next, two works that use AFs to instantiate a KB are presented, explaining in detail how to construct such AFs and what the advantages and disadvantages of each approach are. Table 3.1 can help the reader understand where the current project stands with respect to these other works.

3.1 Knowledge-based Reinforcement Learning

Knowledge bases (KBs) are closely related to traditional GOFAI, since they are central to expert systems, which became very popular in the 80's [21]. A KB can be defined as a collection of expert human information. KBs are useful for knowledge representation (to capture the ontological relations among different concepts) and to capture expert reasoning (i.e., if-then-else rules). In the last decade, there has been an increasing interest in using knowledge-based approaches to improve the interpretability of connectionist ML methods. The survey of Tiddi and Schlobach [56] compiles different works that use knowledge graphs to improve the explanations and common-sense reasoning capabilities of data-driven models.

KB is not only useful to improve interpretability, but it can also boost performance by spotting complicated relations between state features that are too complicated for purely data-driven models to find. Existing works report improved performance in models that benefit from this method of feature extraction [9] [55]. One particular case of knowledge-based reinforcement learning (KBRL) is the work of Voss et al. [57], which uses thousands of rules given by several different human experts to teach an RL agent to play FreeCiv, a strategy game in which players have to build and develop their own empire. The particularity is that, since the rules come from different experts, some of those rules may contradict each other (i.e., two rules may recommend taking a different action in the same situation). As a result, some

TABLE 3.1: Classification of relevant works in KB-RL grouped by model type (connectionist or symbolic). The symbolic approaches are further split according to the conflict resolution method they use.

<table>
<tr><td rowspan="4">KB-RL</td><td>Connectionist</td><td colspan="2">AARL: Gao et al. [24] [23] [22]</td></tr>
<tr><td rowspan="3">Symbolic</td><td colspan="2">Feature-based conflict resolution: FreeCiv [57]</td></tr>
<tr><td rowspan="2">Value-driven conflict resolution</td><td>Model extraction: MARLeME [32]</td></tr>
<tr><td>RL-CO: This project</td></tr>
</table>

conflict-resolution mechanism is needed to execute only one action in the case that such clashes occur.

To illustrate what a rule or "knowledge item" looks like, an example is given in Figure 3.1. Each rule explicitly states its factual knowledge (the raw state features that have to be active for the rule to hold), its working memory (the conditions based on past experience that need to be active for the rule to hold) and the procedural knowledge (the action(s) that the agent should execute if the rule holds).

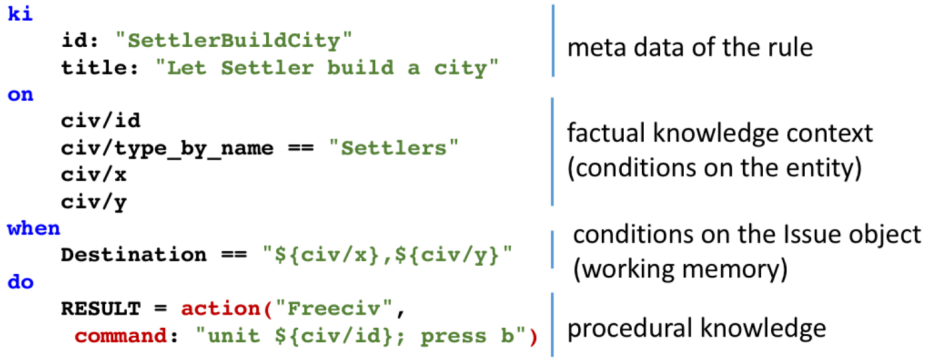


FIGURE 3.1: Example of one of the thousands of knowledge items that comprise the KB in the FreeCiv example. Source: Voss et al. [57].

As mentioned before, conflicting rules must be handled so that only one action is executed. Voss et al. do this by training an RL algorithm that chooses one rule (one of the conflicting knowledge items) given a carefully chosen feature vector made up of 33 game indicators such as game score, population size, amount of generated resources, etc. Nechepurenko et al. [42] compare this KB-RL approach to a neural network to solve a sub-task of the FreeCiv game: to optimise the cities location to maximise the generated natural resources of the player's empire. Overall, the KB-RL yielded results 13% better than the neural-network approach, justifying the injection of expert knowledge to solve the task. Although this solution uses a symbolic approach, the need for a neural network to mediate conflicts among rules conceals important information from the user when a conflict occurs.

In the next section, two works that use argumentation frameworks (AFs) to instantiate a KB are presented. This has the advantage that ties among arguments can be broken using value-driven approaches, allowing for a more transparent conflict resolution.

3.2 Argumentation in Reinforcement Learning

In the last two decades, there has been an increasing interest in applying argumentation to inform machine learning (ML) models with the aim of improving performance, reducing training time and/or for explainability reasons. Cocarascu and Toni [16] offer a survey of different ML approaches that use argumentation.

The first instance of argumentation being applied to an RL problem is found in the work of Gao et al. [24] [23] [22], which focuses on argumentation-accelerated reinforcement learning (AARL). The intuition behind AARL is that, for a particular RL problem, a KB is injected into the model by instantiating an AF. To handle the conflicts that may occur among arguments, some semantics is applied to inform

the RL algorithm about what actions to explore. In this way, the domain expert(s) can recommend actions to the agent while still allowing for random exploration of actions, which can lead to a better policy that may not be captured by the input KB.

While the work of Gao et al. [24] [23] [22] is focused on informing an RL agent during its learning phase, Kazhdan et al. use a model extraction library (MARLeME [32]) to improve the explainability of RL agents by approximating their learnt policy to a symbolic model (a value-based argumentation framework [7]).

Both Gao et al. [24] [23] [22] and Kazhdan et al. [32] use the RoboCup Soccer Simulator environment to implement and evaluate their approaches. The remainder of this section uses the RoboCup Soccer Simulator as our paradigm environment to explain how an AF can be constructed to inform an RL agent on what action to choose and gives further details about the two aforementioned approaches (AARL and model extraction).

3.2.1 Argumentation for Knowledge Instantiation in RL

The first work by Gao et al. [24] to use AA to inform an RL algorithm uses a single-agent setup. In this scenario, one agent has to learn to play Keepaway, a subtask of the popular RoboCup Soccer Simulator[1].

In Keepaway, there are N players whose objective is to keep possession of the ball (the keepers) and N-1 players that try to intercept or tackle the ball (the takers). The game is played inside a fixed rectangular court and the episode finishes when the takers get the ball or when the ball goes off the delimited playing area. The usefulness of Keepaway is that it is a task simple enough to analyse all the possible high-level actions (tackle the ball, mark another player, pass the ball, etc.) of any player at any point in the game, while still offering an interesting and challenging environment.

In this setting, the learning is done exclusively from the perspective of the keeper in possession of the ball. The other players (the keepers not in possession of the ball and the takers) follow hand-coded policies. The work of Gao et al. [24] makes use of the Keepaway framework[2] developed by Stone et al. [52], which provides a series of so-called macro-actions for keepers and takers that allow researchers to abstract away the primitive actions of the players. Such macro-actions are (for the keeper in possession of the ball):

- **HoldBall**(): do not move and keep possession of the ball.
- **PassThenReceive**(k): pass the ball to teammate k and then receive the ball (after the pass, it opens up to receive the ball at a later time). This action calls additional macro-actions $PassBall(k)$ and $Receive()$ [52].

In addition to these macro-actions, some parameters are defined in order to determine the current state of the game, such as the distance from one player to another or the angle between keepers and takers with respect to the keeper who has the ball. Figure 3.2 illustrates this with a snapshot of a game with 3 keepers (K_1, K_2 and K_3) and 2 takers (T_1 and T_2). The keeper with the ball is always referred to as K_1. In this example, we see that θ_i represents the minimum angle between keeper i and any of the takers. Using some threshold parameters L and A, keepers are attributed the status of *far* or *open* when they are at a distance $D \geq L$ and at an angle $\theta \geq$ A from any taker, respectively [24].

[1] https://rcsoccersim.github.io/

[2] https://www.cs.utexas.edu/users/AustinVilla/sim/keepaway/

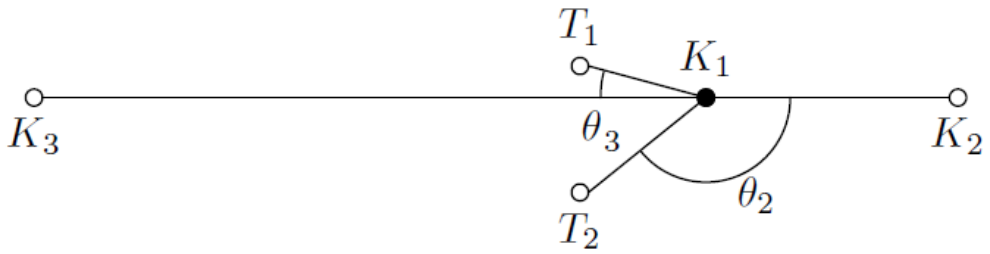


FIGURE 3.2: Example of a Keepaway scenario. In this situation K_3 is not in an open position, since $\theta_3 < A$. Therefore, argument O_3 does not hold under this configuration of the players. Source: Gao et al. [24].

Let us now examine how Gao et al. used argumentation to create an AF to recommend an RL agent which action to take.

Arguments and Actions

Every rule in a KB can be represented in an AF [59]. Gao et al. [24] [23] [22] do this by mapping each rule into an argument whose premises are given by the factual knowledge and working memory of its associated rule. The conclusion of each argument is the procedural knowledge of its corresponding rule. Following this principle, Gao et al. create AFs to recommend actions to RL agents. Its arguments take the follow the structure:

$$con(a) \text{ IF } pre(a) \tag{3.1}$$

where *con(a)*, which is the conclusion of a, recommends an action and $pre(a)$, which is a set of premises, describes the conditions under which a applies (i.e., when in a given situation $pre(a)$ is true, a is a valid argument which recommends the action $con(a)$).

To distinguish between the full AF that considers all the potential arguments that may hold in an environment (e.g., all the arguments that can describe the state of a chess board) and the AF comprised exclusively of the arguments that hold in a particular situation of that environment (e.g., all the arguments that hold in a particular configuration of a chess board), Gao et al. introduce the notion of scenario-specific argumentation framework (SAF) [24]. Analogously, when the AF is value-based, its scenario-specific version will be referred to as a VSAF. A concrete example to illustrate these differences is given next.

RoboCup as a Running Example

Given the set of macro-actions previously mentioned and the specified state variables (being open and/or far with respect to the takers), Gao et al. define the following potential arguments [24]:

- **F_2**: PassThenReceive(2) IF K_2 is *far*
- **O_2**: PassThenReceive(2) IF K_2 is *open*
- **F_3**: PassThenReceive(3) IF K_3 is *far*
- **O_3**: PassThenReceive(3) IF K_3 is *open*
- **H**: HoldBall(): IF *True*

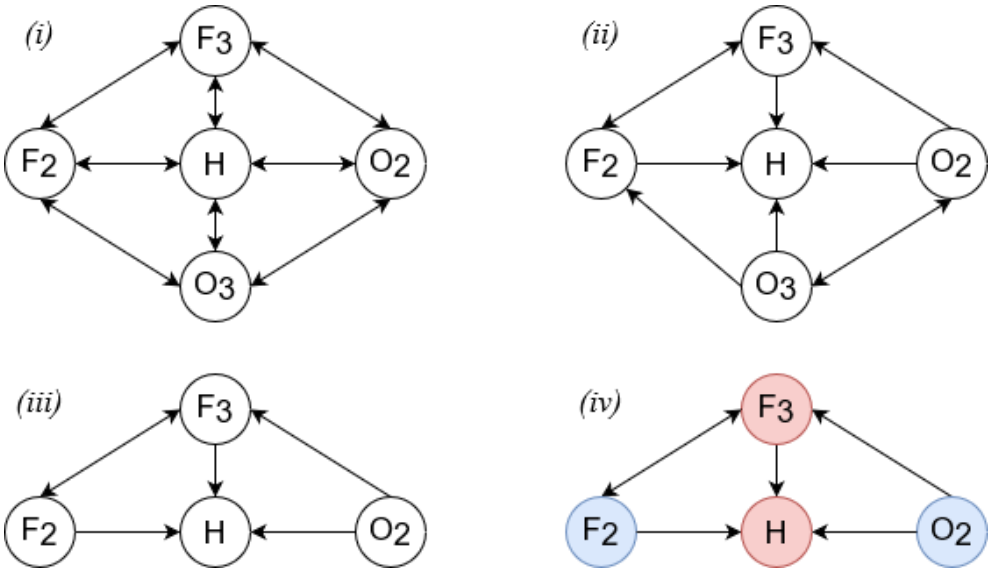


FIGURE 3.3: (i) AF indicating all possible attacks between arguments in Keepaway. (ii) VAF resulting from discarding defeated attacks according to their value. (iii) VSAF composed of the valid arguments from Figure 3.2. (iv) Unique preferred extension (blue nodes). Adapted from Gao et al. [24].

By looking at the actions supported by each argument, it is obvious that arguments supporting different actions must be in an attack relation with each other. Furthermore, from the definition of a VAF given in Section 2.1.1, each argument must promote a value. Namely [24], H promotes the value *"lower the risk of marking"* (MK); O_i promotes the value *"lower the risk of interception"* (IT); and F_i promotes the value *"lower the risk of tackle"* (TK). In addition to creating these arguments and values, the domain expert defines the value preferences for the situation in Figure 3.2 as $IT > TK > MK$.

Let us now demonstrate how argumentation is used by Gao et al. [24] to inject knowledge into an RL agent. The Keepaway situation depicted in Figure 3.2 is used as a running example:

1. **Construct the AF**: a bidirectional arrow is drawn between each two potential arguments (F_3, F_2, O_3, O_2 and H) if they promote different actions (e.g., F_2 and O_2 do not attack each other because both promote *PassThenReceive(2)*). This is illustrated in Figure 3.3(i).

2. **Construct the VAF**: recall that each argument promotes a value and those values are strictly ordered. For every argument a that defeats another argument b (i.e., $Val(a) > Val(b)$), the arrow from b to a is removed from the graph. For instance, in our running example, $Val(O_2) > Val(F_3)$, hence the unidirectional edge from O_2 to F_3 in Figure 3.3(ii).

3. **Construct the VSAF**: the premises of each argument are evaluated according to the current state of the game. If all premises are true, the argument is valid. All the other arguments are invalid. The VSAF is constructed by keeping all the valid arguments and their corresponding bidirectional arrows. As shown in Figure 3.3(iii), this means that in our running example, node O_3 and its incoming/outgoing edges must be removed from the graph.

4. **Calculate the extension**: once the VSAF has been constructed according to the current state of the playing court, a semantics is applied to obtain the extension that contains the accepted arguments under this particular semantics. These arguments are used to recommend an action to K_1. Based on the way the VSAF is constructed, it can be proven that all arguments in each preferred extension recommend the same action [24]. In Figure 3.3(iv), $\{F_2, O_2\}$ is the only preferred extension and both arguments recommend the same action (*PassThenReceive(2)*). Because the grounded semantics always defines a unique extension (it may be the empty set), the grounded semantics is often used to determine a unique recommended action for the learning agent.

The steps above described how Gao et al. used argumentation to create an AF to recommend actions to an RL agent in Keepaway [24]. The next section describes in more detail how they used this technique to guide the exploration of the agent through reward shaping. Section 4 gives an example of an AF constructed for a different game and illustrates how the above-outlined process can be integrated in an RL pipeline.

3.2.2 Argumentation-accelerated Reinforcement Learning

Since the application of argumentation to RL is relatively new, argumentation-accelerated reinforcement learning (AARL) approaches sometimes appear under different names in the literature (e.g., argumentation-based reinforcement learning (ABRL) [24]). In the current work, the term AARL is used in accordance with Cocarascu and Toni [16]: in AARL arguments represent recommendations of actions to individual agents through some extension (e.g., the grounded extension) to guide the exploration process of the agent (i.e., by increasing the chances of visiting state-action pairs the domain expert deems convenient).

One way Gao et al. implemented AARL is by imparting domain knowledge through reward shaping in Keepaway [24]. When implementing this way of reward shaping, the reward function is modified by a *potential-based shaping function* [43] defined as:

$$F(s, a, s', a') = \gamma\Phi(s', a') - \Phi(s, a) \tag{3.2}$$

where $\Phi(s, a)$ is a function that returns a numeric value indicating the *potential* value of being in state s and performing action a. In the approach of Gao et al. [24], this corresponds to the values promoted by the accepted arguments of the AF (i.e., IT, TK or MK). Note the discount factor $\gamma \in (0, 1]$. In the original approach, the learning agent (K_1) learns through a SARSA(λ) with eligibility traces algorithm [54], while it interacts with the other players, which follow hand-coded policies implemented in the Keepaway framework. Equation 3.2 shapes the reward by being added as an extra factor to the reward function. This approach yielded an final reward 5-10% higher compared to the non-accelerated agent (both algorithms were trained for the same amount of time).

Another way Gao et al. used argumentation to guide the exploration of RL agents was by the use of heuristics in a multi-agent setup similar to Keepaway, with the difference that the learning was now done by all the players (both keepers and takers). This subtask of RoboCup was named Keepaway-Takeaway (KATA) [23].

The use of argumentation to guide the exploration in this work [23] slightly differs from that in Keepaway [24], since the the reward function remains now unchanged. The reward shaping is now done by increasing the chances of exploring

specific state-action pairs selected by the domain expert. This is done by modifying the way the RL agent chooses its greedy action at time step t: traditionally, this is chosen according to $argmax_{a_t}[Q(s_t, a_t)]$, where $Q(s_t, a_t)$ measures the quality of choosing action a_t at state s_t. However, this AARL approach incorporates a heuristic function $H_t(s_t, a_t)$, which takes a high value if the (s_t, a_t) pair is recommended by the domain expert. In this case, the greedy action would be chosen according to $argmax_{a_t}[Q(s_t, a_t) + H_t(s_t, a_t)]$.

In the KATA task [23] the state-action pairs recommended by the domain expert are those in which a_t corresponds to the recommended action by the grounded semantics at state s_t. The rationale behind this is that acceptable arguments are more likely to be a plausible action choice for a given state s_t, so the agent would converge faster to an optimal solution if it was more likely to explore these actions suggested by the injected heuristics.

The results in observed in KATA [23] show that argumentation-accelerated RL improved performance with respect to the non-accelerated approach. For the case in which the learning keeper competed against the takers following a hand-coded policy, the final performance score improved an 8% and the convergence time was a 30% faster.

3.2.3 Model Extraction: MARLeME

With the aim of improving the transparency of the policy learnt by RL agents, Kazhdan et al. [32] developed MARLeME, a library for model extraction. MARLeME is designed to approximate the policy learnt by RL models by producing an (approximately) equivalent symbolic model (in this case, a VAF). The authors claim that this approach can be used for interpretability purposes (e.g., to gain insights into how RL models learn by periodically extracting the model during the training phase and analysing how their policy evolves and converges) and for safety purposes (e.g., producing a symbolic model allows for verifiability of the system, thus the original model can be replaced by the symbolic one in scenarios where safety is critical).

In order to translate the original model into a symbolic approximation, MARLeME relies on some domain knowledge that is input by the domain expert through an AF, from which a VAF is derived. MARLeME achieves this by taking the trajectories of one (or several) agent(s) as input and using topological sorting [31] to identify the ordering of the arguments from the given AF that best fit the data. A trajectory is composed of the sequence of states describing an episode along with their corresponding action and reward at each time step.

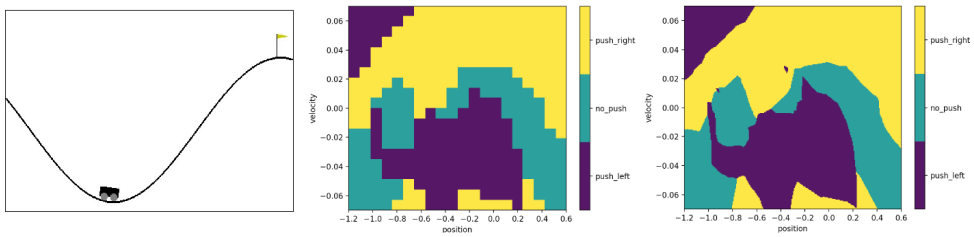
An assumption made by MARLeME about the VAF is that the set of values V is the set of integers, *valpref* is given by the natural ordering of integers, and *val* is a lookup table that assigns an integer to each argument. This assumption is important to find the best VAF without the need for domain knowledge about how each promoted value relates to semantically meaningful concepts, as in the work of Gao et al. [24] (i.e., in their work the domain expert decides that argument H, which supports the action *HoldBall()*, promotes the value "reduce the risk of other keepers being marked". By using integer values to order the arguments, ascribing semantically meaningful concepts to arguments must be done post-hoc by the user after inspecting the trained model).

The described approach was tested by building on the work of Gao et al. [23] to study the Keepaway game exclusively from the perspective of the takers. This specific task is referred to as Takeaway. MARLeME is capable of extracting 3 different VAFs (one for each of the takers), which are easily inspectable. For example, by

looking at the final values, one can interpret that arguments with higher values are more important than those with lower values.

Another approach MARLeME was applied to is the classic Mountain Car problem shown in Figure 3.4a. In this setting, the state is defined by the position and velocity of the car at any given point. Since the state space is continuous, to build an AF like the one shown in Figure 3.3, the state space variables have been discretised, dividing the possible values of position and velocity into 20 different ranges. This gives a total of 400 different states the car can be in at any given point in the game. For each state, a partial AF is active at any given state, consisting of the 3 possible actions (push right, push left, no push) attacking each other. In other words, the AF consists of 1200 arguments, of which only 3 of them hold at any given state and every argument attacks any other argument that recommends a different action.

The result of this approach is shown in Figure 3.4b, where the winning action for each discretised state is shown. The original continuous learnt policy can be found in Figure 3.4c.



a) Mountain car problem b) Extracted Model Policy c) Original Model Policy

FIGURE 3.4: Mountain Car problem (a) along with the extracted (b) and originally learnt (c) model policies. Adapted from Kazhdan et al. [32].

4 Method

At the end of Section 1.2, a four-step overview of the proposed approach was given; namely, (1) designing a game, (2) defining an AF, (3) learning the VAF, and (4) evaluating the VAF. In this chapter, each of these steps is thoroughly explained. Section 4.1 explains the dynamics of Foggy Frozen Lake (FFL), our target game. Two different AFs to solve FFL are proposed in Section 4.2, explicitly stating the premises and conclusions of each of their arguments. Sections 4.3, 4.4 and 4.5 are devoted to explaining how the VAF is learned from the supplied AF just by directly playing the game. Finally, the evaluation criteria are explained in Section 4.6.

4.1 Game

In this thesis, the term *game* will be used to refer to the ultimate task whose reward is expected to be maximised by the VAF we want to learn. For example, in the related work reviewed in Section 3.1, their game was FreeCiv, and in Section 3.2, their game was one of the variants of the RoboCup Soccer Simulator (either Keepaway, KATA, or Takeaway). Similarly, the term *player* will be used to refer to the entity that interacts with the game. Note that the terms *game* and *player* are equivalent to the terms *environment* and *agent*, respectively, traditionally used in the RL paradigm. However, this alternative naming convention will be adopted to prevent misunderstandings in future sections.

To test our proposed approach, a game implemented in the popular RL toolkit Open AI Gym [11] will be used. Open AI Gym is a well-maintained RL toolkit that provides an interface to interact with a variety of games. The game chosen to try our proposed approach is a variant of Frozen Lake.

4.1.1 Foggy Frozen Lake

The original Frozen Lake (FL) is a simple two-dimensional grid-based game set in a virtual world made of ice (this game is provided out-of-the-box with Open AI Gym [11]). The objective in FL is for the agent to move from the *start* cell (S) to the *goal* cell (G) without falling into the *hole* cells (H). To make the game more tangible, a screenshot of FL is shown in Figure 4.1, where the agent is depicted as an elf on an 8x8 grid. The goal is for the elf to find a safe path of *frozen* (F) tiles from his *stool* (in the upper-left tile) to the *present* (bottom-right tile) without falling into the *holes* (H). In the original FL, the observation emitted by the game is an integer corresponding to the index of the tile the agent is at. To allow for the development of strategies when a new map is presented, the game observation is augmented with the (at most) 8 neighbouring tiles of the agent (represented by the red square around the player in Figure 4.1). This is done to allow the player to learn some policy to navigate new maps just by looking at this partial observation of the environment. We call this variant of the game "Foggy Frozen Lake" (FFL). The name comes from the idea that there is some dense fog in the lake, so the agent can only see its surrounding tiles and the index of the tile it is at.

FIGURE 4.1: Screenshot of Foggy Frozen Lake, the game that will be used to test our approach.

Game Dynamics

Each map of FFL is generated according to parameters n and p, where n indicates the size of the grid and p is the probability of a given cell being frozen. When a new game is generated, a squared grid of size nxn is created. Each of its cells is assigned the state "frozen" with probability p or "hole" with probability $1\text{-}p$. If the generated grid world happens to contain no valid paths that connect S to G, a new map is generated until it contains some valid solution. The top left cell is always the starting point and the bottom right cell is always the goal cell. Figure 4.2 illustrates how the complexity of the map varies with different values of p.

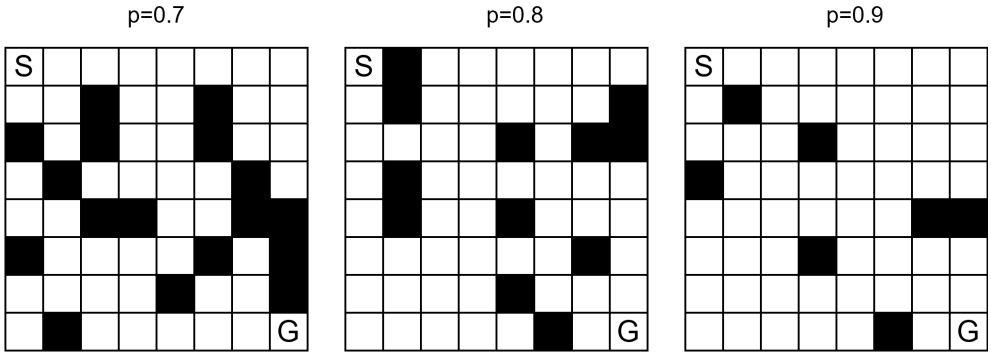


FIGURE 4.2: Example of three 8x8 maps of the FFL game randomly generated for different values of p. S is the starting point of the agent; G is the goal; the black cells represent the holes; and the white cells represent the frozen blocks.

The observations emitted by the game are encoded in a binary array, as shown in Figure 4.3. The first 24 bits encode the tile type of each of the 8 potential tiles surrounding the agent (either a safe tile, a hole, or there is no tile in that position). The encoding is done in 3 groups of 8 bits, which are set to 1 if the tile type corresponds to its group type. Each 8-bit group starts with the left tile with respect to the

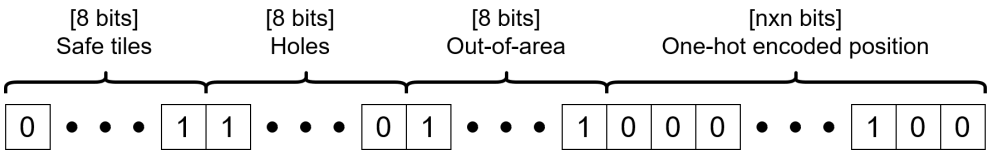


FIGURE 4.3: Structure of the observation emitted by FFL. A binary array encodes the type of all 8 potential neighbouring tiles and the position of the agent in the grid.

agent location and continues clockwise. The remaining bits of the observation are a one-hot encoded vector corresponding to the index of the tile the agent is at. This observation can be directly consumed by a connectionist model or be translated into symbolic features using some KB.

At any state, the agent can perform one of these four actions:

- UP: moves to the cell above.
- DOWN: moves to the cell below.
- RIGHT: moves to the cell on the right.
- LEFT: moves to the cell on the left.

The original formulation of FL assumes that the frozen cells are slippery, so the agent has a probability of $1/3$ of reaching the target cell after choosing an action and a probability of $1/3$ of reaching both perpendicular directions. This stochastic behaviour will be suppressed in our case, meaning that the probability of reaching the target cell will be 1 at all times. If there is no cell in the target direction (e.g., the agent is in the top row and tries to go up), the agent remains in the same state. A reward of 0 is returned in such cases.

The game reaches its terminal state when the agent runs into a hole or when it reaches the goal.

Reward Function

By default, the game returns a unique reward of +1 when the agent reaches the goal state. To help the agent explore successful paths (i.e., actively avoid falling into a hole), the reward function in FFL has been modified so that the game also returns a negative reward in case the agent falls into a hole. Using c_t to refer to the cell class the agent has reached at time step t, the resulting reward function can be defined as:

$$r_t = \begin{cases} -1 & \text{if} \quad c_t = "H" \\ +1 & \text{if} \quad c_t = "G" \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Early Termination

To prevent our agent from roaming indefinitely in the grid, a timeout of 100 time steps is implemented. This means that if the agent does not reach the termination state after 100 steps, the episode is automatically terminated. A reward of 0 is returned in this case.

Additionally, to accelerate the execution time, the episode is also terminated prematurely if the agent performs 2 times in a row the same pair of actions (e.g., UP-DOWN-UP-DOWN). This is a simple way of detecting if the agent is stuck in a loop. This is very common for the symbolic agent in the early stages of the training phase since it is likely for a suboptimal ordering to cause the player to alternate between two actions. In these cases, a reward of 0 is emitted.

4.2 Argumentation Frameworks

This section gives two different AFs to solve the FFL game: a naive AF that uses only factual knowledge (derived purely from the current state) and an advanced AF that also incorporates working memory (derived from previous interactions with the environment). Each AF is given along with its premises (*prem*) and its mapping of arguments to actions (*acts*, as defined in Section 4.5).

4.2.1 A Naive Approach

Let us consider the most basic set of actions that a domain expert could advise the player to take. Since the only primitive action the player can take is to move one cell in one of the four cardinal directions, a simple recommendation could be "move DIRECTION only if that cell is safe", where DIRECTION can be any of $\{up, down, right, left\}$. The set of safe cells is $safe\_cells = \{"S", "G", "F"\}$ (start, goal and frozen). The four possible actions are $UP()$, $DOWN()$, $RIGHT()$ and $LEFT()$. It is assumed that the position of the agent is given by the function $cell(x, y)$. With all these considerations, four different arguments can be created accordingly. The corresponding AF is shown in Figure 4.4 and will be referred to as *naive AF*. The definition of each argument along with its premises and promoted action is as follows:

- **U**: $UP()$ IF $cell(x, y-1) \in safe\_cells$
- **D**: $DOWN()$ IF $cell(x, y+1) \in safe\_cells$
- **R**: $RIGHT()$ IF $cell(x+1, y) \in safe\_cells$
- **L**: $LEFT()$ IF $cell(x-1, y) \in safe\_cells$

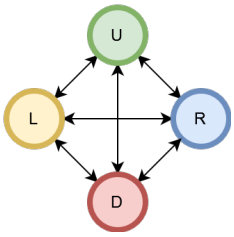


FIGURE 4.4: Naive AF for the FFL game.

4.2.2 Adding Memory

One expert may suggest the player not to move to a tile that has already been explored, to avoid going around in circles. Let $\varphi(x, y)$ be a Boolean function that returns 1 if and only if the agent has not yet been at position (x, y) at some previous time step $t' < t$. The following arguments of the form nX are defined to recommend the player to move to a new tile when that tile is unexplored:

- **nU**: $UP()$ IF $\varphi(x, y+1)$ AND $cell(x, y+1) \in safe\_cells$
- **nD**: $DOWN()$ IF $\varphi(x, y-1)$ AND $cell(x, y-1) \in safe\_cells$
- **nL**: $LEFT()$ IF $\varphi(x-1, y)$ AND $cell(x-1, y) \in safe\_cells$
- **nR**: $RIGHT()$ IF $\varphi(x+1, y)$ AND $cell(x+1, y) \in safe\_cells$

The corresponding AF is given in Figure 4.5 and will be referred to as *advanced AF*. Note that the arrows represent attack relations and all arguments that promote different actions attack each other. Arguments with the same colour promote the same action and do not have an attack relation among them.

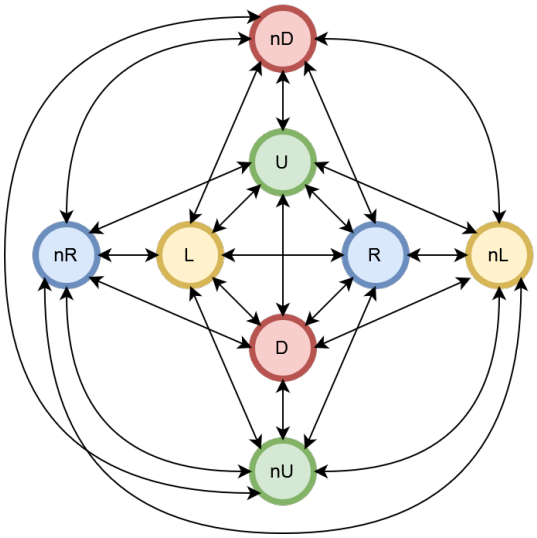


FIGURE 4.5: Advanced AF for the FFL game enriched with information about unexplored surrounding tiles.

4.3 Learning the VAF

As it was mentioned several times throughout Chapter 1, *learning the VAF* amounts to ordering the arguments of the provided AF so that the resulting VAF performs well in the game (in our case, FFL). Since the number of total orderings for an AF of n arguments is $n!$, the solution space quickly becomes too large to be explored using a brute-force approach. As we will see in the next section, this is a particular kind of combinatorial optimisation (CO) problem. Because we want to learn the VAF by directly interacting with the game, some sort of online learning will be necessary. One way to solve this CO problem while performing online learning is through online RL. Using RL to solve CO problems has been a popular research topic in the last few

years, yielding very positive results [6] [33] [4]. A comprehensive survey is done by Mazyavkina et al. [39] where common design decisions in RL-CO are listed (e.g., different ways to encode a problem, different RL algorithms used, etc.). Joshi et al. [30] also explicitly outline the main steps of their CO pipeline.

In this project, an RL-CO pipeline will be used to learn the VAF. The details of how the CO problem is defined are described in Section 4.3.1 and the overall architecture and its components will be explained in Section 4.3.2

4.3.1 Designing the RL-CO Pipeline

The design process of the RL-CO pipeline implemented in this project has been based on the steps outlined in the related literature [39][30]. The five steps followed to design our RL-CO pipeline are:

1. **Problem definition:** given an input AF with $n \in \mathbb{Z}$ nodes, let arg_i denote the i-th argument. Let $\{arg_1, arg_2, ..., arg_n\}$ be a strict total order of the n arguments of the AF where $val(arg_1) > val(arg_2) > ... > val(arg_n)$. The CO problem can be formulated as an MDP, where the state s_t is the partial ordering of the arguments of the input AF up until time step t. Action a_t is the next argument, arg_t, to be appended to the partial ordering at time step t. A VAF can be generated only when s_t is a total ordering (i.e., all arguments are in the partial solution). The VAF is generated from s_t by choosing $valpref$ as the standard ordering of integers and by defining val as an integer mapping function:

$$val : arg \rightarrow i \mid \forall arg \in args(AF), i \in \mathbb{Z} \tag{4.2}$$

   In this equation, $args(AF)$ is the set of arguments of the AF and the integer mapping is done such that the ordering $val(arg_1) > val(arg_2) > ... > val(arg_n)$ holds. The emitted reward is calculated by applying the learnt policy derived from the VAF (as explained in Section 3.2.1) to the target game. The reward signal is zero in all other cases. It can be observed that this problem is formulated as a series of episodic tasks with a fixed number of episodes, so the maximum number of time steps of a full run is always $T = n$.

2. **State encoding:** although tabular methods may be a good approach for small AFs, the exploration space grows factorially as the number of arguments increases. Specifically, the size of the state space is the total number of all partial orderings of the argument set (this is equivalent to the total number of permutations of all subsets of an n-set [51]). This makes the direct exploration of such large state spaces impractical. The proposed solution is to create a set of state features V_{ij} based on the relative ordering of each pair of arguments (i.e., $V_{ab} = 0$ iff $val(arg_a) > val(arg_b)$, otherwise $V_{ab} = 1$). A small example for a 5-argument AF is given in Section 4.4.1 at 3 different time steps.

3. **Action decoding:** the action is iteratively decoded using autoregressive decoding.

4. **Solution search:** the solution is assembled by using greedy search.

5. **Policy learning:** the policy is learnt by using SARSA, an online value-based learning algorithm. The goal is to find an estimate $\hat{q}(s, a)$ of the optimal state-action value function $q_*(s, a)$. To that end, linear function approximation is used to compute $\hat{q}$ from the encoded state features mentioned above. More

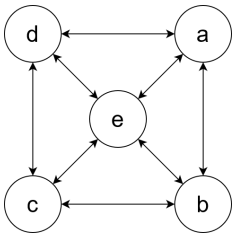


FIGURE 4.6: Example AF with with five arguments (a, b, c, d and e). The arrows indicate attacks between two arguments.

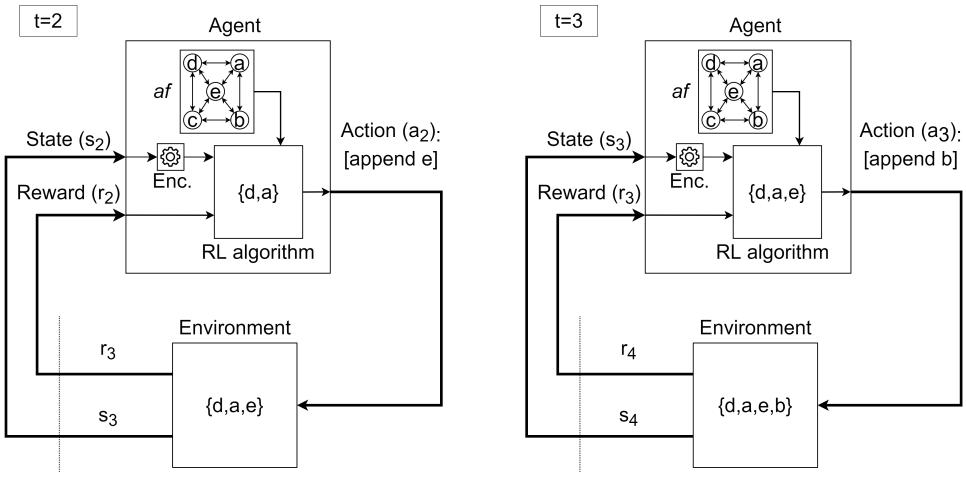


FIGURE 4.7: End-to-end RL-CO pipeline integrating an argumentation framework at time step $t = 2$ (left) and $t = 3$ (right). When $t < T - 1$ the partial solution does not need to be evaluated on the game (contrast with Figure 4.8).

details about linear function approximation and SARSA are given in Section 4.4.2.

4.3.2 Pipeline Architecture and Components

To better illustrate this process, a schematic representation of a concrete architecture is now given, using the 5-argument AF shown in Figure 4.6 as the input domain knowledge to our running example. Figure 4.7 illustrates how a partial solution is constructed by showing the pipeline evolution from time step $t = 2$ to $t = 3$. Keep in mind that the first action was taken at time step $t = 0$ (i.e., according to the figure, argument d was selected by a_0).

Here is a breakdown of each of the elements that appear in Figure 4.7:

- **Agent:** the entity that learns the policy of some target game by finding an optimal ordering of the arguments of *af* (the input AF). It will be described in detail in Section 4.4.
- **State s_t:** the partial ordering of the 5 arguments up until time step t.
- **Reward r_t:** the reward at time step t. If $t = T - 1 = 4$ and s_t contains no duplicate arguments (all 5 arguments have been sorted), r_t is equal to the reward given by the target game after evaluating the learnt policy. Otherwise $r_t = 0$.

- ***af*:** the expert knowledge input by the domain agent in the form of an AF.
- **Enc. (encoding):** encoded representation of the current state: the partial ordering is transformed into a set of state features.
- **RL algorithm:** algorithm that learns $\hat{q}(s, a)$: a semi-gradient SARSA with linear function approximation will be used. This block outputs the next action.
- **Action a_t:** the next argument to be appended to the partial permutation.
- **Environment:** the state of the environment of the formulated MDP is the partial permutation of the 5 arguments resulting from adding the last argument to it. This block is a simplification of the full environment since the policy is not evaluated in the target game until $t = T - 1$. A full representation of the environment dynamics is given in Figure 4.8. Its dynamics will be described in detail in Section 4.5.

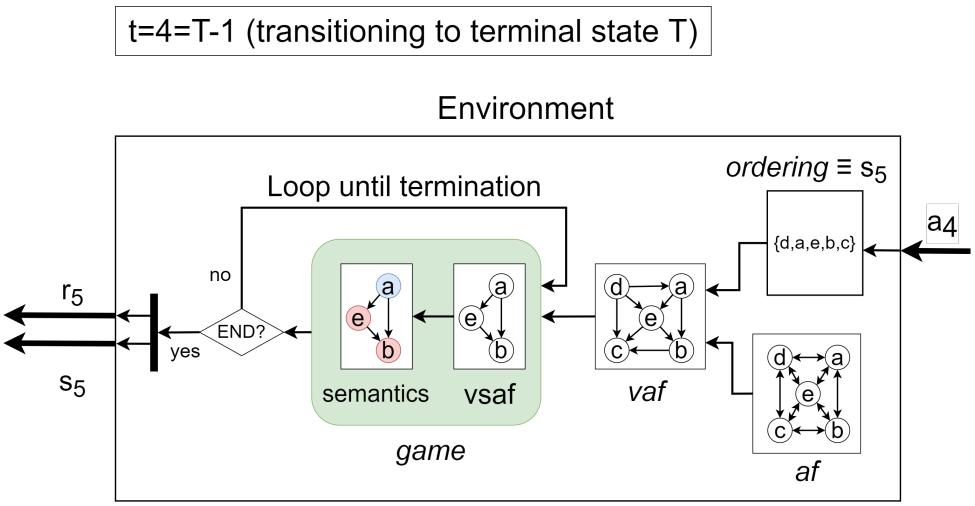


FIGURE 4.8: Detailed diagram of the environment dynamics: once s has become a total ordering, the derived policy is evaluated on the *game* instance and both the reward and the final ordering are returned.

As mentioned in the *problem definition*, the policy is not evaluated on the target game until time step $t = T - 1$. For this reason, and to simplify the pipeline representation, the *environment* block in Figure 4.7 has omitted all the details regarding how the AF is used in conjunction with the *val* function to determine the policy of the target game. The workflow at time step $t = T - 1$ (being $t = T$ the time step of the terminal state T), is shown in a separate diagram in Figure 4.8. The specific constituent blocks of the *environment* block are described below:

- **Ordering:** block representing both the *val* function and the $valpref$ relation of the learnt VAF. It is the ordering of the arguments in *af* once all arguments have been strictly sorted, being the first element the one with the highest value and the last one the argument with the lowest value (as stated in the *problem definition*). It is equivalent to the last state of the environment (i.e., in Figure 4.8, the *val* function is derived from s_5, since at this point all 5 arguments have been ordered), and $valpref$ is the standard integer ordering (see Section 4.3.1).
- ***af*:** same AF as in the *agent* block.

- ***vaf*:** resulting VAF after removing the defeated attacks from *af*, as described in Section 3.2.1.
- ***game*:** the target game for which a policy is being learnt. The policy is only evaluated once all arguments have been ordered. To evaluate the policy, the game is played according to the policy defined by vaf. This is done by iteratively computing a VSAF and its corresponding extension to decide on the next action to take inside *game* (e.g., Figure 4.8 depicts a snapshot of the target game where the VSAF is composed of arguments a, e and b). This is done iteratively until some *termination* condition is met (hence the "loop until termination" arrow pointing back at the start). Ideally, the *game* would reach some terminal state and return some reward. However, some early termination strategies will be implemented to accelerate the execution, such as detecting strategies that make the agent loop between two states.
- ***vsaf*:** derived VSAF, as described in Section 3.2.1.
- ***semantics*:** an extension is calculated according to some semantics to recommend an action to take inside *game*. The grounded extension is used, as in the related work [23] [32].

4.4 Agent

The previous section has translated the problem of learning the VAF into a CO problem and the overall RL-CO pipeline has been explained. This section describes the agent used to solve the CO problem of finding the best ordering of the arguments of the AF. First, the encoding used to represent the state of the environment is described. Then, the learning algorithm is outlined. Finally, the reward constraints imposed on the action space of this agent are described.

4.4.1 State Encoding

As seen in Section 4.3.2, the observation emitted by our environment is a (partial) ordered set. This observation cannot be readily used by our agent, so a set of features has to be constructed from it. Since the observation informs the agent about the relative order of the arguments, the features should explicitly express this information. One way to encode this is by creating a binary squared matrix of size $|args(af)|$ (the number of arguments in *af*), where each value in the matrix represents the relative ordering between two arguments.

Let n be the number of arguments in *af*. Let $(s, \succ)$ be a strict (partial) ordered set over the arguments of af. Let M be a squared binary matrix of size $n \times n$, where arg_k denotes the k-th argument of $args(af)$. Each element $m_{ij} \in M$ is defined according to:

$$m_{ij} = \begin{cases} 0 & \text{if} \quad val(arg_i) > val(arg_j) \\ 1 & \text{otherwise} \end{cases} \tag{4.3}$$

Note that Equation 4.3 has been designed in such a way that all possible state encodings yield a non-zero matrix to ensure that at least some features are active at all times. This is important when using linear methods as the learning algorithm.

Using the environment depicted in Figure 4.8 as a running example, Table 4.1 illustrates the different values of M at time steps $t = 0$, $t = 2$ and $t = 5$. It is

easy to verify that M biunivocally encodes the order relations of s. For example: in Table 4.1(i), the element in position (1,2) is 1 because it is not true that $val(b) > val(c)$, according to s_0; in Table 4.1(ii), the element in position (0,1) is 0 because $val(a) > val(b)$, according to s_2 (although b has not been ordered yet, we know that arguments added later have lower order than their predecessors); and in Table 4.1(iii), the element in position (0,3) is 1 because $val(a) < val(b)$, according to s_5.

TABLE 4.1: Proposed encoding at different time steps for the environment from Figure 4.8. Matrix indices start at 0.

(i) $s_0 = \{\}$

	a	b	c	d	e
a	1	1	1	1	1
b	1	1	1	1	1
c	1	1	1	1	1
d	1	1	1	1	1
e	1	1	1	1	1

(ii) $s_2 = \{d, a\}$

	a	b	c	d	e
a	1	0	0	1	0
b	1	1	1	1	1
c	1	1	1	1	1
d	0	0	0	1	0
e	1	1	1	1	1

(iii) $s_5 = \{d, a, e, b, c\}$

	a	b	c	d	e
a	1	0	0	1	0
b	1	1	0	1	1
c	1	1	1	1	1
d	0	0	0	1	0
e	1	0	0	1	1

4.4.2 Learning Algorithm

For the learning algorithm, a value-based method will be used. As seen in Section 2.2.2, the goal in value-based methods consists in finding a function $q_\pi(s, a)$ that approximates q_*, the optimal action-value function. To do so, linear function approximation [54] will be used to estimate q_π and its parameters will be updated according to the Q-learning algorithm [58].

Linear Function Approximation

The estimate $\hat{q}(s, a)$ will be calculated using linear function approximation [54]. This approach uses a set of weights, $\mathbf{W}$, and a feature matrix, $\mathbf{X}(s, a)$, to approximate the values of the state-action pair (s, a).

The feature matrix $\mathbf{X}$ is a binary matrix of dimensions $\mathcal{S} \times \mathcal{A}$, where $\mathcal{S}$ are the dimensions of the matrix encoding of the encoded, M, and $\mathcal{A}$ is the cardinality of the set of actions. In the FFL case, the size of $\mathbf{X}$ is $|args(af)| \times |args(af)| \times 4$, since there are 4 possible actions. The value of the element x_{ijk} is 1 when the encoded state feature m_{ij} is active and when the current action a corresponds to the k-th possible action. In every other case, x_{ijk} is 0.

The weights matrix $\mathbf{W}$ has the same dimensions as $\mathbf{X}$. The goal is to find a $\mathbf{W}$ that yields a good approximation of $\hat{q}_\pi$ according to:

$$\hat{q}(s, a, \mathbf{W}) \doteq \mathbf{W} : \mathbf{X}(s, a) = \Sigma_{ij}(w_{ijk} x_{ijk}) \tag{4.4}$$

where $(\cdot):(\cdot)$ denotes the Frobenius inner product.

The next section explains how to update the weights of $\mathbf{W}$ to approximate the optimal value function q_*. For that, the gradient of $\hat{q}_\pi$ with respect to $\mathbf{W}$ will be necessary, which is simply:

$$\nabla \hat{q}(s, a, \mathbf{W}) \doteq \mathbf{X}(s, a) \tag{4.5}$$

SARSA

SARSA [48] is an on-policy learning algorithm that uses the action at the next state to approximate the expected return of the next state (also known as bootstrapping). Assuming $\hat{q}$ is a linear function, as defined in the previous section, the SARSA update rule is:

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \alpha\big(r_t + \gamma\hat{q}(s_{t+1}, a_{t+1}, \mathbf{W}_t) - \hat{q}(s_t, a_t, \mathbf{W}_t)\big)\nabla\hat{q}(s, a, \mathbf{W}_t) \tag{4.6}$$

4.4.3 Constraining the action space

Early experiments were conducted to test the efficacy of the agent. Although successful, some preliminary tests revealed that the random exploration of the agent becomes very inefficient as the number of arguments in *af* increases. This is because as the number of arguments to be ordered increases, it becomes more likely for a clash to occur between arguments already in the partial solution and an argument chosen at random. For example, for a 5-argument AF, the probability of randomly choosing the penultimate argument without clashing is $2/5$; however, for a 15-argument this probability is $2/15$.

For this reason, the action space of the agent at any time step t is limited to the set of arguments that are not yet in the partial solution s_t. This approach is consistent with similar work done in the RL-CO domain [34] [12].

4.5 Environment

Several details of the environment have already been explained in Section 4.3.1. This section gives further details about its dynamics. Our environment emulates the CO problem of finding and ordering for the arguments of the AF fed by the domain expert such that the resulting VAF maximises the accumulated rewards emitted by the game. The environment needs 4 parameters to be instantiated:

- *game*: an instance of the target game, as described in Section 4.1.
- *af*: the AF supplied by the domain expert
- *prem*: the premises for each of the arguments in *af*. This is essentially a mapping function that, given an observation, outputs the list of valid arguments.
- *act*: a function that maps every argument in *af* with the action it promotes.

Our environment has been conveniently implemented using the Open AI Gym [11] toolkit.

4.5.1 Environment Dynamics

The state of the environment is given by the (partial) ordering of the arguments of *af*. After the environment has been initialised, the initial state (s_0) is an empty set and the possible set of actions is the set of arguments of *af*. This means that at every time step t, one of these actions can be appended to the current state. To illustrate this, a visual representation of this process is given in Figure 4.9. In this diagram, it can be seen that the environment receives an action a_t (for example, argument d) and appends it to the partial solution, which corresponds to the new state of the

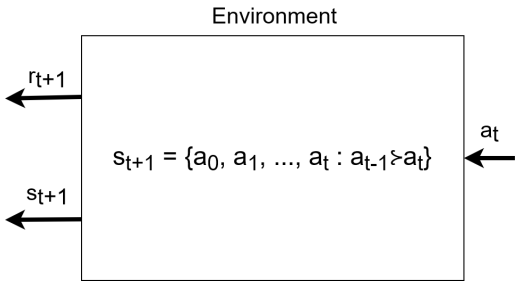


FIGURE 4.9: Simplified diagram of the environment: state s_t evolves at time step t when an action a_t is selected.

environment. More formally, we can say that the state of the environment, s, is a strict partial ordering, $(s, \succ)$, over the arguments of *af*.

Once every argument of *af* is in the ordering, s becomes a strict total ordering and the terminal state is reached since there is no argument left to order. At this point, s represents the *ordering* of our VAF, so vaf can now be created by removing the unsuccessful attacks from *af*, as we saw in Section 3.2.1.

After the VAF has been constructed, the derived policy is evaluated on the *game* instance. To do so, a VSAF and its corresponding extension are calculated at every step of the *game* block. The applied semantics determine the extension whose arguments get accepted, as outlined in Section 3.2.1. The action promoted by the accepted arguments (all arguments of the grounded extension promote the same action) is applied in the *game* instance until it reaches the terminal state. At that point, the environment outputs its current state and the accumulated reward signal emitted by *game*. This process is conceptually illustrated in Figure 4.8. For example, if the input *af* is the AF from Figure 4.6, the current ordering of the arguments is given by the total ordering $\{d, a, e, b, c\}$. Since all arguments are ordered, the vaf is calculated and the target *game* is evaluated. At the current time step inside the *game* block, only arguments a, b and e are valid, according to the input *prem* function. According to the semantics, the grounded extension is $\{a\}$, so the action promoted by argument a, is applied in the *game* block at that particular instant. The *game* evolves and the next $vsaf$ is calculated until the *game* reaches its terminal state.

4.5.2 Reward Function

The only point at which the policy can be evaluated on the *game* is once all the arguments of *af* have been ordered. At any other time step, the reward returned by the environment is 0. Let *args(af)* denote the set of argument of an AF. Let *rew(game, vaf, prem, act)* denote the accumulated reward emitted by the input *game* instance given the derived VAF (*vaf*), the premises of its arguments (*prem*) and the mapping of arguments to their promoted action (*act*). The reward function can be defined as:

$$r_t = \begin{cases} rew(game, vaf, prem, act) & \text{if} \quad args(af) \cap s_t = args(af) \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

4.6 Evaluation

In order to assess our approach, multiple criteria will be taken into account; namely, interpretability, absolute performance and domain generalisation. To have a better

idea of the quality of our solution, the results are compared to various baseline models. This section contains an overview of the different baseline agents our approach has been compared to and details about the metrics of interest that will be the focus of our experiments in Chapter 5.

4.6.1 Baseline models

The symbolic agents (SA) will be compared to the following baseline agents.

Non-symbolic agent (NSA)

The non-symbolic agent (NSA) takes the raw observation emitted by the game, as shown in Figure 4.3, as state features. Similar to our proposed approach, it uses the SARSA learning algorithm to select the best possible action a for any given state s_t and linear function approximation to implement $\hat{q}(s, a, \mathbf{W})$ to estimate the value of each state-action pair.

Random agent (RA)

The random agent (RA) is an agent that samples one of the possible actions from a uniform distribution regardless of the state of the world.

Surroundings-aware Random Agent (SRA)

The surroundings-aware random agent (SRA) is analogous to the RA, with the exception that the actions that would lead the agent to fall into a hole have been removed from the set of possible actions.

Handcrafted-policy Agent (HA)

A simple strategy consists in prioritising actions that make the agent move towards the goal (right or down) provided that the corresponding cell is safe. If the agent can safely move either right or down, the corresponding actions are added to a set of candidate actions. Otherwise, the remaining actions are added to the candidate set provided that they lead to a safe cell. The handcrafted-policy agent (HA) uses these criteria to create a set of candidate actions and randomly samples one at each time step of the game. The corresponding algorithm is given in Appendix A.

4.6.2 Metrics

Three different aspects can be examined to answer Research Question 1; namely, interpretability, absolute performance and domain generalisation. These three aspects are in direct relation to the secondary Research Questions 2, 3 and 4, respectively.

Interpretability

Coming back to the definition by Biran and Cotton [8] given in Chapter 1 (*"[...] systems are interpretable if their operations can be understood by a human [...]"*), the interpretability of our model will be assessed qualitatively by examining the final ordering of the arguments of the AF. For example, in the FFL game, since the goal is always at the Southeast of the starting point, it would be expected that the found policy prioritises the actions down() and right(). This is analogous to the strategy

analysis done with MARLeME [32]: in the Takeaway game in which MARLeME was evaluated, the authors looked at the arguments with the highest values for each of the takers to produce a post-hoc interpretation of the policy.

Success Rate

The success rate will be used as a proxy to measure absolute performance (the total reward obtained by an agent). This is justified because the FFL is a win-or-lose game, so the success rate is a more informative metric. To measure the success rate, the learning agents (naive/advanced SA and NSA) will be trained until convergence and they will be evaluated using their greedy policy (i.e., the RL agent no longer explores at this stage). All agents will be trained each run on a a fixed game instance and the results will be averaged across runs.

Success Rate: Domain Generalisation

To assess if injecting domain knowledge in the agent improves its success rate on novel scenarios, the agents will be evaluated on game instances sampled from distributions different from the one used during the training phase.

5 Experiments

In this chapter, two experiments performed with the agents are shown. The code to run the experiments is publicly available[1] for replication along with the implementation of all the agents and environments used here. The two symbolic agents will be referred to as *naive SA* and *advanced SA*, implementing the naive AF and advanced AF, respectively (see Section 4.2).

5.1 Baseline Comparison: One Game at a Time

In this experiment, the three learning agents (naive/advanced SA and NSA) are trained on a single instance of FFL per run, sampled from a distribution with parameters $n = 8$ and $p \in \{0.6, 0.7, 0.8, 0.9\}$. In each run, four different game instances are generated (one for each possible value of p) and the learning agents are trained until some convergence criterion is met (specifically, until the agent solves the game 100 times in a row following its greedy policy) for a maximum of 5000 episodes. After training, the learning agents are evaluated according to their greedy policy. The other three (non-learning) baseline models (RA, SRA and HA) are also evaluated on each FFL instance. A total of 50 runs are executed (i.e., 50 game instances for each value of p) and the results are averaged across runs. The more expressive the state features, the better is the model expected to perform. For example, the NSA has access to the tile index, so it should be able to explore every tile and find a safe path without much effort. On the other hand, the arguments of the naive SA rely exclusively on the relative position of the agent with respect to its neighbouring safe/unsafe cells. The advanced SA is more expressive than the naive SA because it also "knows" whether the neighbouring cells are unexplored or not.

5.1.1 Results

The success rate of each model per value of p is shown in Table 5.1. The NSA was capable of learning every environment it was presented with, as we would expect since it is the most expressive model. The advanced SA performed almost at par with the NSA, however, the lower score for $p = 0.6$ shows that the advanced AF is not expressive enough to solve all game instances. The effect of adding memory to the advanced SA is clear when comparing it to the performance of the naive AF (ranging between 0.38 and 0.82), which performed worse than the HA (ranging between 0.46 and 0.88). The very low success rates of the RA and the SA show that the game is not trivial. The next experiment will focus on comparing the NSA and the advanced SA, using only the HA for reference.

[1] https://github.com/omcandido/RL-AA

TABLE 5.1: Average success rate obtained by each model for different values of p.

model	**p**			
	0.6	**0.7**	**0.8**	**0.9**
RA	0.00	0.00	0.00	0.00
SRA	0.04	0.02	0.06	0.12
HA	0.46	0.72	0.74	0.88
naive SA	0.38	0.36	0.66	0.82
advanced SA	0.98	1.00	1.00	1.00
NSA	1.00	1.00	1.00	1.00

5.2 Towards a Universal Strategy: Domain Generalisation

In this experiment, the game instance is re-sampled at each episode from a distribution with parameters $n = 8$ and $p = 0.8$ and the NSA and the advanced SA are trained extensively until policy convergence. To evaluate the domain generalisation capabilities, each model is evaluated (using its greedy policy) on a new game instance sampled from a distribution with variable parameters n and p. Specifically, for each run, 14 different game configurations are created by combining the elements of the sequences $(0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9)$ and $(8, 16)$ for parameters p and n, respectively. The results are averaged across 1000 runs. In this case, the advanced SA is expected to outperform the NSA, since the advanced SA benefits from state features derived from domain knowledge that help capture its latent state (similar to Nechepurenko et al. [42]). A good target function is expected to be too complex for our NSA model to approximate since linear function approximation predictions are based exclusively on the current observations of the game and do not have an internal state that can capture the latent dynamics of the POMDP. Neural architectures that exhibit some "memory" capabilities thanks to their internal state (such as RNNs) are common connectionist choices to solve POMDPs [13] [40].

5.2.1 Results

At the end of the training, the ordering of the advanced SA converged to:

$$\{nR, nD, L, U, nL, nU, D, R\} \tag{5.1}$$

The average success ratio for each model at every game configuration is shown in Figure 5.1. Interpretability and performance are discussed separately in the following sections.

Domain Generalisation

The second observation is that the success ratio decreases as the map size and complexity increase, regardless of the agent. The results show that the SA can consistently find the safe path to the goal even when the game is not sampled from the same distribution as the training instances. Specifically, Figure 5.1 shows that the SA achieved an average success ratio higher than 90% when trained on a distribution of games sampled with $n = 8$ and $p = 0.8$. Without any further training, the same model is capable of solving less complicated games ($n = 8, p = 0.9$) with an average success ratio of nearly 100%, and more complicated games ($n = 8, p = 0.6$) with an

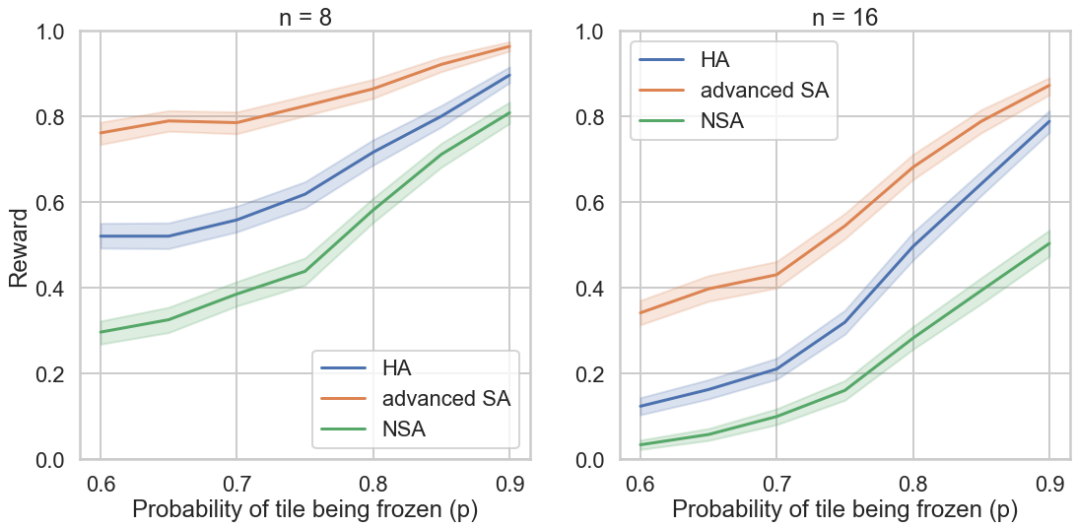


FIGURE 5.1: Visualisation of how the advanced SA and the NSA generalise to new game distributions after being trained only on instances with parameters *n=8* and *p=0.8*. The HA performance is also shown for reference.

average success ratio above 80%. It can also be seen that the SA agent scales reasonably well when the map width is doubled ($n = 16, p = 0.8$), reaching an average success ratio of above 80%. The agent still scales reaches a success ratio of almost 100% when the complexity of the game is decreased ($n = 16, p = 0.9$), but this time it does not fare so well for more complex games ($n = 16, p = 0.6$), where the success ratio drops to nearly 50%. It can be seen that, for each of the game configurations, the SA performs significantly better than the HA, which in turn performs better than the NSA. As mentioned in the previous section, this low performance of the NSA is most likely due to not being able to integrate past interactions with the game into its new inferences, and relying exclusively on the current observation of the game. It may seem surprising that the HA, which does not leverage past experiences either, performs better than the NSA. This is thanks to the fact that the HA breaks ties at random between its preferred actions, allowing it to perform different actions in the same tile and avoid getting stuck in a loop.

Interpretability: Strategy Analysis

Regarding the interpretability of the model, the advanced SA produced the following ordering: $\{nR, nD, L, U, nL, nU, D, R\}$. This means that moving right and down, if the tiles are new and safe, are the two most important arguments (nR and nD, respectively). Next, going left and up, if the tiles are just safe, are the following two preferred arguments (L and U), respectively). The fact that the first and second arguments are antagonists of the third and the fourth allows the agent to trace back its steps in case it enters a cul-de-sac (e.g., if the agent moves to the right, as indicated by its first argument, and enters a corridor with no exit, the argument nR will no longer hold since those tiles have already been explored and the agent will fall back to argument L, allowing it to retrace its steps and look for an alternative). The next arguments in the ordering are nL and nU. These two arguments are superseded by L and U, respectively, since they have less constrained premises, higher order of preference and recommend the same action. So, if nL holds, L will also hold and will

be preferred. If L does not hold, nL will not hold either, because $pre(L) \subset pre(nL)$. Finally, D and R are the least preferred arguments. This makes sense because any of them being the only argument of the grounded extension would imply that the agent has already explored the direction they indicate to explore (e.g., if nR holds, R also holds, but nR will be preferred. If R holds but nR does not hold, it is because the agent has already explored its right tile). This means that when the agent defaults to any of the last two arguments, it will inevitably loop the same sequence of actions over and over again. Figure 5.2 shows the traces of an agent following the aforementioned policy. It can be seen that the overall strategy is to sweep the map from left to right and from the top to the bottom until the goal is reached.

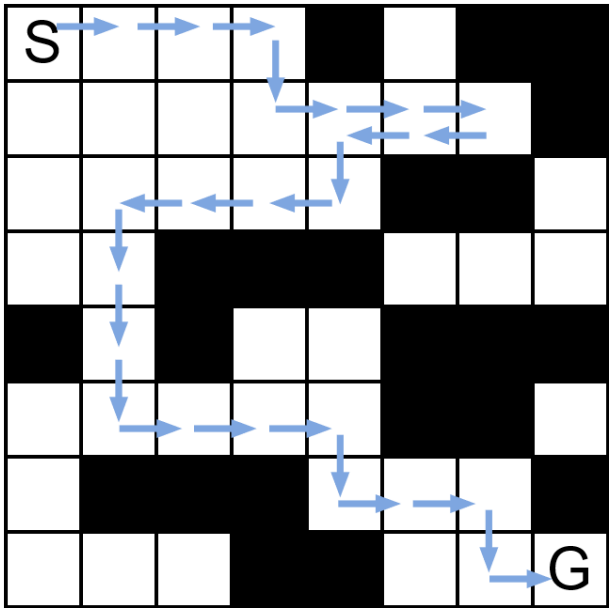


FIGURE 5.2: Policy determined by a VAF with ordering $\{nR, nD, L, U, nL, nU, D, R\}$ on a FFL map generated with n=8 and p=0.6.

6 Discussion

This chapter summarises the contributions of this thesis to the scientific literature on KB-RL. The advantages and disadvantages of the method described in Chapter 4 are discussed and some pointers for potential future lines of research are given.

6.1 Conclusion

This project presented a novel approach to training a fully symbolic RL agent using formal argumentation with the aim of improving the interpretability of AI-made decisions. To justify the proposed approach, the rationale for some scholars [25] to advocate more interpretable models has been explained and three relevant works on RL have been described in detail [57] [22] [32]. Although model-extraction techniques [32] are a promising approach, the fact that the resulting symbolic model is an approximation of the original RL agent leaves open the possibility that the extracted model may provide unfaithful explanations or perform worse than the original agent. To avoid these problems, an end-to-end ML pipeline has been proposed to bypass the model-extraction step and train a symbolic agent using online RL. The resulting symbolic agent uses a VAF as its inference engine, which results in better explanations than connectionist conflict resolution techniques (e.g., compare to Voss et al. [57]).

The implementation of the proposed approach is explained in detail in Chapter 4, along with a very simple strategy game to test it. The experiments from Chapter 5 show that our pipeline can be used to train a symbolic model to solve a variety of games when provided an AF rich enough. In view of these results alone, it can be said that Research Question 1 has been positively answered. The remainder of this section analyses the extent to which the secondary research questions were answered.

6.1.1 Explainability

Coming back to the definition of explainability by Biran and Cotton [8] given in Chapter 1, our model can be considered explainable by virtue of its introspectability. Specifically, Section 5.2 shows how the learnt strategy can be explained by examining the final ordering of the arguments, giving a positive answer to Research Question 2. This kind of post-hoc analysis was also done by Kazhdan et al. [32] with the VAFs extracted by MARLeME, so it is fair to wonder what the added value of our approach is compared to using model extraction to replace the original model. First, let us consider using MARLeME to periodically extract a symbolic model from a connectionist RL agent running in production. This flexible approach might help understand what the agent is doing, but, because faithfulness cannot be guaranteed between the explanations and the deployed models, this method is not suitable for applications that require the model to be verifiable. Now, let us consider the scenario in which the model extracted by MARLeME is used to replace the original

model in production. Because the extracted model is an approximation, some performance loss is likely to occur (the authors report a fidelity score of their models varying between 0.86 and 0.68 in RoboCup Takeaway). If the performance of the extracted model is acceptable, (i.e., the accumulated reward is high enough for the application), then model extraction would be a viable solution. However, if the quality of the model deteriorates too much to be deployed, either a new agent needs to be trained on the original task (in hopes that its new trajectories will be captured better by the same AF) or a more complex AF needs to be fed into MARLeME (with the expectation that it will fit better the trajectories of the agent). In any case, the fundamental problem has shifted from solving a certain RL problem to imitating the behaviour of another agent. This means that the trained agent is acting as a surrogate for the original task. Thus, although the resulting VAF contains the ordering that best fits the trajectories of the agent, there is no guarantee that this is the ordering that would give the highest rewards when interacting with the environment. This is because, as mentioned in Section 1.2, $\Pi_{traj} \subseteq \Pi_{env}$, being Π_{traj} and Π_{env} the policy space when considering training on the trajectories of the model and the state-action-reward space of the environment , respectively.

6.1.2 Performance

As a secondary goal, the performance of our symbolic model was compared with multiple baseline models (most interestingly, a non-symbolic model and a handcrafted model) to get a sense of how feature extraction from domain knowledge can improve model performance. Section 5.1 showed that the performance of the naive SA could be improved by creating a richer AF (the advanced AF). As a result, the advanced SA performed almost at par with the NSA. While the NSA solved 100% of the game instances, still some game instances could not be solved by the advanced SA. The fact that a simple handcrafted policy could achieve relatively high performance as well, is a likely sign that the FFL game was too simple to categorically answer Research Question 3. What can be said is that the NSA was expressive enough to learn a good policy each time and that the performance of the naive naive SA could be increased by enriching its AF. Therefore, the absolute performance of our symbolic agent is bounded by the expressiveness of its AF.

Section 5.2 presented a more challenging task. This time, the agents were trained on a distribution of game instances instead of a single instance at a time. In this scenario, the advanced SA performs much better than any other model and shows notable generalisation capabilities when evaluated on instances sampled from novel distributions. Our baseline NSA is using linear function approximation, which does a poor job at capturing the latent state of the game (e.g., visiting a cell for the second time may have certain implications). Since memory played a fundamental role in improving the performance of the symbolic agent, a recurrent neural network (RNN) would likely improve the generalisation capabilities of our NSA. Regarding Research Question 4, our experiments do not constitute a comprehensive comparison among symbolic and non-symbolic agents, but a simple game such as FFL already presents a somewhat challenging function for a non-symbolic agent. It is shown that, in our setup, symbolic agents can leverage expert domain knowledge to learn a solution that can be applied to novel environments.

6.2 Future Work

The choice of an AF in the style of Dung [17] was motivated by the works of Gao et al. [22] and Kazhdan et al. [32], who also use the same kind of AF to instantiate domain knowledge. An interesting line of research could investigate the impact of different AFs on the interpretability of the symbolic agent. For example, imagine that an expert on FFL creates the following generic argument: "*if the agent has already performed action X in the current tile, then it should now perform an action Y such that $Y \neq X$*". Instantiating this domain knowledge in our AF would require 12 additional arguments (pUL, pUR, pUD, pDL, pDR, pDU, etc.), where each argument pXY means that a *previous* action X has been performed in the current tile and action Y is now recommended. The full AF is shown in Figure 6.1(i) and this was in fact one of the first AFs fed to our symbolic agent. Although the results were positive in terms of performance, using a long set of abstract arguments made the resulting ordering difficult to interpret. Consider now a bipolar argumentation framework (BAF) [2] that extends Dung's definition of AF adding a support relation between arguments. Instead of a formal definition, a visual example is given in Figure 6.1(ii) translating the aforementioned AF. By allowing support and attack relations among arguments, only 4 new arguments are needed this time (each argument pXY can be conflated into a single pX argument with one attack relation and three support relations). Using this technique, a default set of arguments can promote the primitive actions in the action space (i.e., in this case, U, D, L and R) while new arguments are added attacking/supporting other arguments without necessarily promoting an action. This is an interesting approach that presents new challenges, such as designing a new conflict resolution mechanism (either value-based [15] or weight-based [45]) and choosing a semantics that ensures that exactly one of its arguments promotes a primitive action.

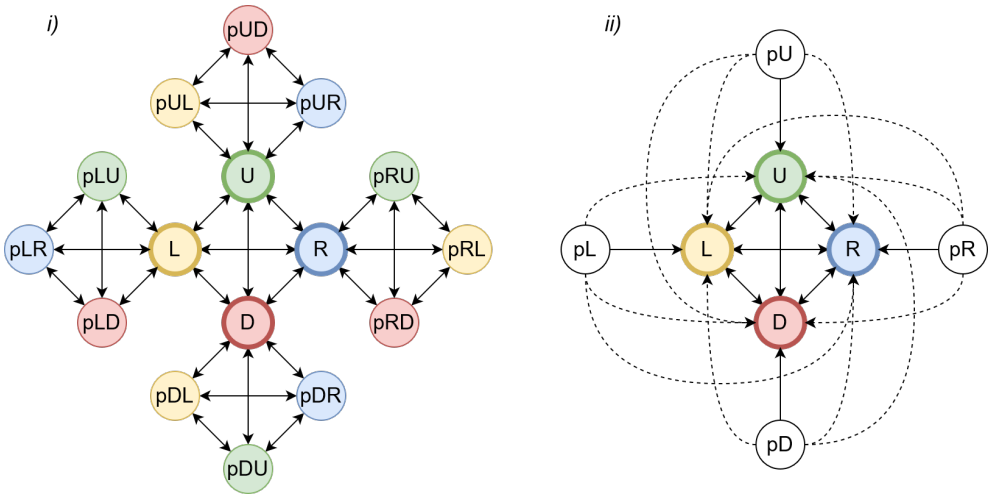


FIGURE 6.1: Comparison between (i) an AF in the style of Dung [17] and (ii) the equivalent BAF [2]. Some attacks have been omitted for simplicity in the AF on the left (arguments with different colors are in attack relation with each other). On the right, the attack relations of the arguments in the BAF are represented with a continuous arrow, while the support relations are represented with a dashed arrow. Arguments with a white background do not promote any action.

As mentioned in the previous section, this project constitutes by no means a rigorous comparison between symbolic and non-symbolic agents. One line of research that goes in the direction of performance comparison should consider both a more challenging game and a more sophisticated architecture for the non-symbolic agent. Since the literature using a KB to solve an RL task is very limited, there is no standardised test that can be used as a common benchmark by multiple researchers. Such a benchmark should provide both an RL environment and a KB. The KB can be given in the form of a set of rules that can be instantiated using some formalism such as an AF.

Finally, finding a good ordering for the arguments of our AF has been formulated here as a CO problem and RL was proposed to solve it due to the factorial growth of the state space with the number of arguments in the AF. This is a difficult problem on its own and the solution offered here is subject to optimisation. For example, games like FFL return very sparse rewards, slowing down the learning process. Modest speed improvements have been achieved a posteriori using replacing traces [54] in our RL algorithm. Other methods such as memory replay or the use of deep learning architectures can potentially improve the convergence time and reward of our symbolic RL agent. Additionally, other alternatives can be explored to solve the CO problem formulated in Section 4.3.1 (e.g., genetic algorithms).

A Handcrafted Agent

Algorithm 1 Handcrafted-policy Agent Algorithm

```
a ← {}                                   ▷ Initialise the set of candidate actions
if RIGHT is safe then
    a ← right()
end if
if DOWN is safe then
    a ← down()
end if
if a is empty then
    if LEFT is safe then
        a ← left()
    end if
    if UP is safe then
        a ← up()
    end if
end if
Randomly sample an action from a
```

References

[1] Ahn, Woo kyoung et al. "The role of covariation versus mechanism information in causal attribution". In: *Cognition* 54.3 (1995), pp. 299–352. ISSN: 0010-0277. DOI: https://doi.org/10.1016/0010-0277(94)00640-7. URL: https://www.sciencedirect.com/science/article/pii/0010027794006407.

[2] Amgoud, Leila, Cayrol, Claudette, and Lagasquie-Schiex, Marie-Christine. "On the bipolarity in argumentation frameworks". In: *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, Canada, June 6-8, 2004, Proceedings*. Ed. by James P. Delgrande and Torsten Schaub. 2004, pp. 1–9. URL: http://www.pims.math.ca/science/2004/NMR/papers/paper01.pdf.

[3] Baroni, Pietro, Caminada, Martin, and Giacomin, Massimiliano. "An introduction to argumentation semantics". In: *The Knowledge Engineering Review* 26.4 (2011), pp. 365–410. DOI: 10.1017/S0269888911000166. URL: https://doi.org/10.1017/S0269888911000166.

[4] Barrett, Thomas D. et al. "Exploratory Combinatorial Optimization with Reinforcement Learning". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 3243–3250. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5723.

[5] Bellman, Richard. "On the theory of dynamic programming". In: *Proceedings of the national Academy of Sciences* 38.8 (1952), pp. 716–719.

[6] Bello, Irwan et al. "Neural Combinatorial Optimization with Reinforcement Learning". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=Bk9mxlSFx.

[7] Bench-Capon, Trevor J. M. "Value Based Argumentation Frameworks". In: *CoRR* cs.AI/0207059 (2002). URL: https://arxiv.org/abs/cs/0207059.

[8] Biran, Or and Cotton, Courtenay. "Explanation and justification in machine learning: A survey". In: *IJCAI-17 workshop on explainable AI (XAI)*. Vol. 8. 1. 2017, pp. 8–13.

[9] Borghesi, Andrea et al. "Injective domain knowledge in neural networks for transprecision computing". In: *International Conference on Machine Learning, Optimization, and Data Science*. Springer. 2020, pp. 587–600.

[10] Brandão, Rafael et al. "Mediation Challenges and Socio-Technical Gaps for Explainable Deep Learning Applications". In: *CoRR* abs/1907.07178 (2019). arXiv: 1907.07178. URL: http://arxiv.org/abs/1907.07178.

[11] Brockman, Greg et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[12] Cappart, Quentin et al. "Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization". In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 3677–3687. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16484.

[13] Carr, Steven, Jansen, Nils, and Topcu, Ufuk. "Verifiable RNN-based policies for POMDPs under temporal logic constraints". In: *arXiv preprint arXiv:2002.05615* (2020).

[14] Caruana, Rich et al. "Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-day Readmission". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. Ed. by Longbing Cao et al. ACM, 2015, pp. 1721–1730. DOI: 10.1145/2783258.2788613. URL: https://doi.org/10.1145/2783258.2788613.

[15] Cayrol, Claudette and Lagasquie-Schiex, Marie-Christine. "Gradual Valuation for Bipolar Argumentation Frameworks". In: *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 8th European Conference, ECSQARU 2005, Barcelona, Spain, July 6-8, 2005, Proceedings*. Ed. by Lluís Godo. Vol. 3571. Lecture Notes in Computer Science. Springer, 2005, pp. 366–377. DOI: 10.1007/11518655_32. URL: https://doi.org/10.1007/11518655_32.

[16] Cocarascu, Oana and Toni, Francesca. "Argumentation for Machine Learning: A Survey". In: *Computational Models of Argument - Proceedings of COMMA 2016, Potsdam, Germany, 12-16 September, 2016*. Ed. by Pietro Baroni et al. Vol. 287. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, pp. 219–230. DOI: 10.3233/978-1-61499-686-6-219. URL: https://doi.org/10.3233/978-1-61499-686-6-219.

[17] Dung, Phan Minh. "On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games". In: *Artif. Intell.* 77.2 (1995), pp. 321–358. DOI: 10.1016/0004-3702(94)00041-X. URL: https://doi.org/10.1016/0004-3702(94)00041-X.

[18] Elton, Daniel C. "Self-explaining AI as an Alternative to Interpretable AI". In: *Artificial General Intelligence - 13th International Conference, AGI 2020, St. Petersburg, Russia, September 16-19, 2020, Proceedings*. Ed. by Ben Goertzel et al. Vol. 12177. Lecture Notes in Computer Science. Springer, 2020, pp. 95–106. DOI: 10.1007/978-3-030-52152-3_10. URL: https://doi.org/10.1007/978-3-030-52152-3_10.

[19] Ewerth, Ralph et al. ""Are Machines Better Than Humans in Image Tagging?" - A User Study Adds to the Puzzle". In: *Advances in Information Retrieval*. Ed. by Joemon M Jose et al. Cham: Springer International Publishing, 2017, pp. 186–198. ISBN: 978-3-319-56608-5.

[20] Feng, Shi et al. "Pathologies of Neural Models Make Interpretations Difficult". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 3719–3728. DOI: 10.18653/v1/D18-1407. URL: https://aclanthology.org/D18-1407.

[21] Frankish, Keith and Ramsey, William M. *The Cambridge handbook of artificial intelligence*. Cambridge University Press, 2014.

[22] Gao, Yang. "Argumentation accelerated reinforcement learning". PhD thesis. Imperial College London, UK, 2014. URL: http://hdl.handle.net/10044/1/26603.

[23] Gao, Yang and Toni, Francesca. "Argumentation Accelerated Reinforcement Learning for RoboCup Keepaway-Takeaway". In: *Theory and Applications of Formal Argumentation - Second International Workshop, TAFA 2013, Beijing, China, August 3-5, 2013, Revised Selected papers*. Ed. by Elizabeth Black, Sanjay Modgil, and Nir Oren. Vol. 8306. Lecture Notes in Computer Science. Springer, 2013, pp. 79–94. DOI: 10.1007/978-3-642-54373-9_6. URL: https://doi.org/10.1007/978-3-642-54373-9_6.

[24] Gao, Yang, Toni, Francesca, and Craven, Robert. "Argumentation-Based Reinforcement Learning for RoboCup Soccer Keepaway". In: *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*. Ed. by Luc De Raedt et al. Vol. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012, pp. 342–347. DOI: 10.3233/978-1-61499-098-7-342. URL: https://doi.org/10.3233/978-1-61499-098-7-342.

[25] Gerlings, Julie, Shollo, Arisa, and Constantiou, Ioanna D. "Reviewing the Need for Explainable Artificial Intelligence (xAI)". In: *54th Hawaii International Conference on System Sciences, HICSS 2021, Kauai, Hawaii, USA, January 5, 2021*. ScholarSpace, 2021, pp. 1–10. URL: https://hdl.handle.net/10125/70768.

[26] Goodman, Bryce and Flaxman, Seth. "European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation"". In: *AI Magazine* 38.3 (2017), pp. 50–57. DOI: 10.1609/aimag.v38i3.2741. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/2741.

[27] Hahn, Ulrike, Bluhm, Roland, and Zenker, Frank. *Causal argument*. Oxford, UK: Oxford University Press, 2017.

[28] Hinkelmann, Knut, Ahmed, Sajjad, and Corradini, Flavio. "Combining Machine Learning with Knowledge Engineering to detect Fake News in Social Networks - A Survey". In: *Proceedings of the AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE 2019) Stanford University, Palo Alto, California, USA, March 25-27, 2019., Stanford University, Palo Alto, California, USA, March 25-27, 2019*. Ed. by Andreas Martin et al. Vol. 2350. CEUR Workshop Proceedings. CEUR-WS.org, 2019. URL: http://ceur-ws.org/Vol-2350/paper10.pdf.

[29] Jacovi, Alon and Goldberg, Yoav. "Towards Faithfully Interpretable NLP Systems: How Should We Define and Evaluate Faithfulness?" In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Ed. by Dan Jurafsky et al. Association for Computational Linguistics, 2020, pp. 4198–4205. DOI: 10.18653/v1/2020.acl-main.386. URL: https://doi.org/10.18653/v1/2020.acl-main.386.

[30] Joshi, Chaitanya K. et al. "Learning TSP Requires Rethinking Generalization". In: *CoRR* abs/2006.07054 (2020). arXiv: 2006.07054. URL: https://arxiv.org/abs/2006.07054.

[31] Kahn, Arthur B. "Topological sorting of large networks". In: *Commun. ACM* 5.11 (1962), pp. 558–562. DOI: 10.1145/368996.369025. URL: https://doi.org/10.1145/368996.369025.

[32] Kazhdan, Dmitry, Shams, Zohreh, and Liò, Pietro. "MARLeME: A Multi-Agent Reinforcement Learning Model Extraction Library". In: *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207564. URL: https://doi.org/10.1109/IJCNN48605.2020.9207564.

[33] Khalil, Elias et al. "Learning combinatorial optimization algorithms over graphs". In: *Advances in neural information processing systems* 30 (2017).

[34] Kool, Wouter, Hoof, Herke van, and Welling, Max. *Attention, Learn to Solve Routing Problems!* 2018. DOI: 10.48550/ARXIV.1803.08475. URL: https://arxiv.org/abs/1803.08475.

[35] Li, Zongbo, Jiao, Zaibin, and He, Anyang. "Knowledge-based artificial neural network for power transformer protection". In: *IET Generation, Transmission & Distribution* 14.24 (2020), pp. 5782–5791. DOI: https://doi.org/10.1049/iet-gtd.2020.0542. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-gtd.2020.0542. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-gtd.2020.0542.

[36] Liao, Beishui, Anderson, Michael, and Anderson, Susan Leigh. "Representation, justification, and explanation in a value-driven agent: an argumentation-based approach". In: *AI Ethics* 1.1 (2021), pp. 5–19. DOI: 10.1007/s43681-020-00001-8. URL: https://doi.org/10.1007/s43681-020-00001-8.

[37] Maat, Emile de, Krabben, Kai, and Winkels, Radboud. "Machine learning versus knowledge based classification of legal texts". In: *Legal Knowledge and Information Systems*. IOS Press, 2010, pp. 87–96.

[38] Marcus, Gary. "The next decade in AI: four steps towards robust artificial intelligence". In: *arXiv preprint arXiv:2002.06177* (2020).

[39] Mazyavkina, Nina et al. "Reinforcement learning for combinatorial optimization: A survey". In: *Computers & Operations Research* 134 (2021), p. 105400. DOI: 10.1016/j.cor.2021.105400. URL: https://doi.org/10.1016/j.cor.2021.105400.

[40] Meng, Lingheng, Gorbet, Rob, and Kulić, Dana. "Memory-based deep reinforcement learning for pomdps". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 5619–5626.

[41] Miller, Tim. "Explanation in artificial intelligence: Insights from the social sciences". In: *Artif. Intell.* 267 (2019), pp. 1–38. DOI: 10.1016/j.artint.2018.07.007. URL: https://doi.org/10.1016/j.artint.2018.07.007.

[42] Nechepurenko, Liudmyla, Voss, Viktor, and Gritsenko, Vyacheslav. "Comparing Knowledge-Based Reinforcement Learning to Neural Networks in a Strategy Game". In: *Hybrid Artificial Intelligent Systems - 15th International Conference, HAIS 2020, Gijón, Spain, November 11-13, 2020, Proceedings*. Ed. by Enrique A. de la Cal et al. Vol. 12344. Lecture Notes in Computer Science. Springer, 2020, pp. 312–328. DOI: 10.1007/978-3-030-61705-9_26. URL: https://doi.org/10.1007/978-3-030-61705-9_26.

[43] Ng, Andrew Y., Harada, Daishi, and Russell, Stuart J. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In: *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*. Ed. by Ivan Bratko and Saso Dzeroski. Morgan Kaufmann, 1999, pp. 278–287.

[44] Oestermeier, Uwe and Hesse, Friedrich W. "Singular and general causal arguments". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 23. 23. 2001.

[45] Pazienza, Andrea, Ferilli, Stefano, and Esposito, Floriana. "Constructing and Evaluating Bipolar Weighted Argumentation Frameworks for Online Debating Systems". In: *Proceedings of the 1st Workshop on Advances In Argumentation In Artificial Intelligence co-located with XVI International Conference of the Italian Association for Artificial Intelligence, AI^3@AI*IA 2017, Bari, Italy, November 16-17, 2017*. Ed. by Stefano Bistarelli, Massimiliano Giacomin, and Andrea Pazienza. Vol. 2012. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 111–125. URL: http://ceur-ws.org/Vol-2012/AI3-2017_paper_12.pdf.

[46] Pollock, John L. "Defeasible Reasoning". In: *Cognitive Science* 11.4 (1987), pp. 481–518. DOI: 10.1207/s15516709cog1104_4. URL: https://doi.org/10.1207/s15516709cog1104_4.

[47] Ribeiro, Marco Túlio, Singh, Sameer, and Guestrin, Carlos. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. Ed. by Balaji Krishnapuram et al. ACM, 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778.

[48] Rummery, G. A. and Niranjan, M. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. TR 166. Cambridge, England: Cambridge University Engineering Department, 1994.

[49] Shen, Shiwen et al. "An interpretable deep hierarchical semantic convolutional neural network for lung nodule malignancy classification". In: *Expert Systems with Applications* 128 (2019), pp. 84–95. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2019.01.048. URL: https://www.sciencedirect.com/science/article/pii/S0957417419300545.

[50] Silver, David et al. "Mastering the game of Go without human knowledge". In: *Nat.* 550.7676 (2017), pp. 354–359. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

[51] Sloane, N. J. A. *Entry A000522 in the On-Line Encyclopedia of Integer Sequences*. https://oeis.org/A000522.

[52] Stone, Peter, Sutton, Richard S., and Kuhlmann, Gregory. "Reinforcement Learning for RoboCup-Soccer Keepaway". In: *Adaptive Behavior* 13.3 (2005), pp. 165–188.

[53] Strasser, Christian and Antonelli, G. Aldo. "Non-monotonic Logic". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2019. Metaphysics Research Lab, Stanford University, 2019.

[54] Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*. MIT press, 2018.

[55] Teng, Teck-Hou and Tan, Ah-Hwee. "Knowledge-Based Exploration for Reinforcement Learning in Self-Organizing Neural Networks". In: *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. Vol. 2. 2012, pp. 332–339. DOI: 10.1109/WI-IAT.2012.154.

[56] Tiddi, Ilaria and Schlobach, Stefan. "Knowledge graphs as tools for explainable machine learning: A survey". In: *Artificial Intelligence* 302 (2022), p. 103627. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2021.103627. URL: https://www.sciencedirect.com/science/article/pii/S0004370221001788.

[57] Voss, Viktor et al. "Playing a Strategy Game with Knowledge-Based Reinforcement Learning". In: *SN Comput. Sci.* 1.2 (2020), p. 78. DOI: 10.1007/s42979-020-0087-8. URL: https://doi.org/10.1007/s42979-020-0087-8.

[58] Watkins, C. J. C. H. "Learning from Delayed Rewards". PhD thesis. King's College, Oxford, 1989.

[59] Wyner, Adam Zachary, Bench-Capon, Trevor J. M., and Dunne, Paul E. "Instantiating Knowledge Bases in Abstract Argumentation Frameworks". In: *The Uses of Computational Argumentation, Papers from the 2009 AAAI Fall Symposium, Arlington, Virginia, USA, November 5-7, 2009*. Vol. FS-09-06. AAAI Technical Report. AAAI, 2009. URL: http://aaai.org/ocs/index.php/FSS/FSS09/paper/view/940.

[60] Zhang, Quanshi and Zhu, Song-Chun. "Visual interpretability for deep learning: a survey". In: *Frontiers Inf. Technol. Electron. Eng.* 19.1 (2018), pp. 27–39. DOI: 10.1631/FITEE.1700808. URL: https://doi.org/10.1631/FITEE.1700808.

[61] Zhou, Kaiyang et al. "Domain generalization: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).