Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 270 (2025) 1232–1241

Procedia Computer Science

www.elsevier.com/locate/procedia

29th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2025)

Towards a neurosymbolic approach based on Anticipatory Learning Classifier Systems and Ontologies

Romain Orhand[a,c], Ali Ayadi[b,c]

[a]*MLMS team, Icube Laboratory - UMR 7357, 300 bd Sébastien Brant, F-67412, Illkirch, France*
[b]*SDC team, Icube Laboratory - UMR 7357, 300 bd Sébastien Brant, F-67412, Illkirch, France*
[c]*University of Strasbourg, 4 rue Blaise Pascal, F-67081, Strasbourg, France*

Abstract

This paper aims to achieve a system in which the learning of Anticipatory Learning Classifier Systems (ALCS) is guided by ontologies, allowing us to benefit from the advantages of each approach. ALCS are powerful rule-based models for decision-making in complex, dynamic, and partially observable environments. While their symbolic nature offers strong adaptability, their inductive learning process can lead to invalid or unsafe decisions in domains requiring expert knowledge, constraints, or safety guarantees. To reach our goal, (1) ALCS should be capable of solving increasingly complex learning tasks and (2) a framework enabling us to make ALCS interact with ontologies is needed. We first propose to remove a learning bias present in ALCS. We demonstrate through extensive experiments that comparable performance on standard maze benchmarks is achieved by ALCS without this bias while enabling, for the first time, the solving of the cart-pole problem. We then introduce a novel neurosymbolic approach in which ALCS are coupled with ontologies acting as semantic safeguards. Through symbolic reasoning, the ontology can validate, reject, or suggest alternative actions based on its encoded knowledge, enhancing safety, guiding exploration, and improving interpretability. These contributions pave the way toward knowledge-guided, explainable, and human-in-the-loop learning systems.

Keywords: Machine learning; Reinforcement learning; Anticipatory learning classifier systems; Ontologies; Neurosymbolic approach

1. Introduction

Anticipatory Learning Classifier Systems (ALCS) are powerful models for learning and decision-making in complex, dynamic, and partially observable environments [6]. ALCS build populations of condition–action–effect rules (called classifiers), which are learned from the agent's interactions with its environment by trying to anticipate the consequences of their actions according to their current situations [10]. Thus, these systems learn a representation

E-mail addresses: rorhand@unistra.fr (Romain Orhand)., ali.ayadi@unistra.fr (Ali Ayadi).

1877-0509
10.1016/j.procs.2025.09.244

of their environment from which they learn a decision-making policy aimed at solving their learning task through reinforcement learning.

The symbolic nature of ALCS facilitates interpretability and knowledge integration. Since their rules can be read, reasoned upon, or modified, they are ideal candidates for integration with symbolic knowledge sources such as ontologies. This opens the door to hybrid neurosymbolic architectures whose learning is guided and validated by domain knowledge, improving safety for instance [15].

Our motivation is to achieve a system in which the learning of an ALCS is guided by an ontology, allowing us to benefit from the advantages of each approach. It then presupposes (1) ALCS capable of solving increasingly complex learning tasks (2) as well as a framework enabling us to make ALCS interact with ontologies. ALCS are able to solve complex tasks in non-deterministic partially observable environments in which their partial observations do not allow them to distinguish distinctly the situations in which they are (that is the perceptual aliasing issue), or in which the outcome of their actions may be uncertain [20]. Surprisingly, however, ALCS are unable to solve classic problems in the reinforcement learning literature, such as the cart-pole problem [12].

This paper is consequently our first step towards designing such a neurosymbolic approach, by enabling ALCS to solve new reinforcement learning tasks such as the cart-pole problem and presenting the first steps of a framework in which ontologies guide ALCS learning.

The rest of the paper is organized as follows: section 2 introduces the main principles of ALCS and a summary of research works related to ALCS. Then, a learning bias that may impair the learning capabilities of ALCS is studied in section 3. Section 4 focuses on the classic cart-pole problem: we propose an explanation about why this problem has never been solved by any ALCS, which is related to the previous learning bias, along with a solution to enable its resolution by these systems. Our approach to couple ALCS with ontologies is presented in section 5, before concluding in section 6.

2. Principles of Anticipatory Learning Classifier Systems

ALCS learn classifiers that consist at least of a condition, an action, an effect, as well as internal parameters such as experience, quality and reward prediction [22]. Condition and effect components are made of symbols that describe perceived environmental changes following an action given a set of situations. A wildcard can be used to indicate that an element can take on all possible values in the condition components, or to indicate that no change has been perceived in the effect components. Classifier experience gives the number of times a classifier has been used by the ALCS. Classifier quality is a measure of how accurate anticipations are. Classifier reward prediction indicates expected rewards if the classifier is used given a learning task.

Conditions, actions, effects and qualities of ALCS classifiers are learned through the Anticipatory Learning Process, that compares perceptions retrieved successively to detect differences between them, which would be due to the actions carried out by ALCS [10]. Reinforcement Learning is used to update predictions of expected rewards and adapt classifiers to learning tasks.

The ALCS generic learning cycle is that of ACS2 from which are derived the main ALCS used today: they perceive new sensory inputs, manage their population of classifiers thanks to Anticipatory Learning Process, Reinforcement Learning or other processes, and carry out actions from the set of classifiers matching the sensory inputs according to an action selection policy. For further details on the learning cycle of ACS2, we suggest readers to read [7]. Other processes may enable ALCS to build more general classifiers thanks to genetic algorithms [3] or to detect non-deterministic environmental properties to build classifiers able to deal with them [5, 16, 18, 19, 20].

Other ALCS-related studies have focused on: setting up efficient exploration bias to speed up building of environmental representations [4]; adapting action planing to ACS2 [24]; applying averaged reward criterion in multi-step environments [13]; learning of ACS2 in discretized real-valued environments [12, 14]; compacting efficiently ALCS populations of classifiers [17]; designing an experience replay extension to ALCS that enables the replay of entire episodes [21].

Among all existing ALCS, we have chosen to use BEACS for the rest of the paper [20]. BEACS is derived from ACS2 where new processes to handle non-determinism have been modularly designed. Those processes can be then activated or deactivated at will to make BEACS behave like ACS2 or other similar ALCS. Moreover, those processes enable BEACS to efficiently deal with much more complex environments that will be more similar to our real-life environments and learning tasks that will be of interest to us in future research works.

3. Debiasing without impairing learning of ALCS

The first anticipatory learning classifier system ACS originally does not care about classifiers that predict no environmental changes [22]. When most ALCS now learn such classifiers, their learning cycle still places greater emphasis on classifiers or actions that lead to environmental changes. This is the learning bias we then discuss.

3.1. Removing an old bias

The learning cycle of ACS2, from which are derived the main ALCS used today such as BEACS, promotes classifiers whose effect components anticipate changes in two places [7]:

1. Reinforcement learning uses the discounted maximum payoff predicted in the next time-step that is computed from the set of classifiers that match the current situation and do anticipate environmental changes.
2. If the most promising action has to be chosen, the ϵ-greedy policy used to select actions only considers classifiers anticipating environmental changes.

Removing this bias is quite straightforward as it only implies in the code to update the conditions related to the computation of the discounted payoff and to the ϵ-greedy policy by removing a test on classifier effect components. The whole code can be found at https://github.com/RomainOr/Reasoning-with-ALCS along with jupyter-notebooks used for experimentations.

3.2. Experimental protocol and metrics

The mazes can be easily manipulated to create all kinds of situations or to have all kinds of properties we want to investigate. This is the reason why those environments are traditionally used as testbeds within the literature on anticipatory learning classifier systems to study their learning capabilities. Using the maze benchmark and the metrics from [20], the debiased version of BEACS (named d-BEACS in the rest of the article) is compared with BEACS in an experimental protocol set to address the following question: **Does the learning bias affect BEACS performance in mazes?**

The benchmark is made up of 23 mazes of different complexities, for instance due to the presence of indistinguishable states based solely on current perception. The results of actions have a 25% chance of being uncertain, in which case the system performs a random action without knowing which one. The use of random action makes learning more complex and enables us to analyze whether the bias affects BEACS ability to deal with non-deterministic environments. The goal of agents in these mazes is both to build a complete and accurate representation of their environment and reach the exit as fast as possible. Agents can move in mazes one grid cell at a time, in either eight adjacent positions. Their perception is limited to the eight squares adjacent to each position. Their starting position in mazes is random but distinct from the exit.

For each maze of the benchmark, 30 runs were performed using each of these two versions of BEACS. A run firstly consists of a succession of 7000 trials, that are constrained explorations until the exit or the maximum number of actions (100) are reached: ϵ is set to 0.8 for the ϵ-greedy policy used to select actions; the learning rate of the Anticipatory Learning Process and Reinforcement Learning, β, is set to 0.05; the maximal number of actions in classifiers is set to 3. Then, the ALCS are switched to pure exploitation (*i.e.* without Anticipatory Learning Process). They have 1000 trials to bootstrap an efficient decision policy ($\epsilon = 0.2, \beta = 0.05$), 1000 more trials to stabilize rewards ($\epsilon = 0, \beta = 0.05$), and finally 1000 more trials where the average number of actions required to reach the exit is recorded ($\epsilon = 0, \beta = 0.05$). Other ALCS-related parameters not described here are initialized to the default values provided in [20].

The following metrics were also collected and averaged over the 30 runs, for each environment: classifier population sizes, knowledge ratios, and average EP-accumulated errors (EP stands for Enhanced Predictions). The knowledge ratio is the ratio of the correct transitions learned by at least one reliable classifier to all possible transitions. Only transitions that led to environmental changes are considered. The average EP-accumulated error characterizes the environmental fitness of the probabilities of occurrence of transitions learned by classifiers and is thus complementary

Maze	Population sizes		Knowledge ratios (%)		Average EP-errors (%)		Steps to exit	
	BEACS	d-BEACS	BEACS	d-BEACS	BEACS	d-BEACS	BEACS	d-BEACS
Cassandra4x4	1614(198)	1647(258)	99.35(0.89)	99.09(0.97)	4.11(0.28)	4.13(0.26)	3.52(0.16)	3.55(0.20)
Littman57	515(52)	504(60)	100.00(0.00)	99.77(0.87)	2.48(0.32)	2.68(0.56)	6.02(0.51)	6.10(0.54)
Littman89	912(92)	904(76)	99.83(0.45)	99.96(0.24)	3.51(0.37)	3.36(0.23)	5.88(0.40)	6.00(0.59)
Maze10	755(67)	712(62)	100.00(0.00)	99.90(0.55)	2.66(0.92)	2.21(0.66)	18.02(12.76)	28.28(22.04)
Maze4	231(29)	222(4)	99.97(0.16)	100.00(0.00)	1.50(0.19)	1.45(0.15)	4.80(0.08)	4.79(0.09)
Maze5	301(4)	299(5)	100.00(0.00)	99.95(0.17)	1.52(0.11)	1.49(0.05)	6.32(0.09)	6.34(0.09)
Maze7	172(24)	166(19)	100.00(0.00)	100.00(0.00)	1.00(0.12)	1.00(0.14)	5.85(0.28)	5.89(0.26)
MazeA	176(0)	176(0)	100.00(0.00)	100.00(0.00)	1.12(0.04)	1.11(0.04)	5.92(0.09)	5.92(0.08)
MazeB	1410(142)	1360(144)	99.90(0.31)	99.80(0.55)	3.64(0.19)	3.54(0.24)	6.05(0.56)	5.90(0.52)
MazeD	726(76)	704(49)	99.79(0.42)	99.79(0.42)	3.04(0.16)	3.12(0.25)	4.21(0.22)	4.28(0.25)
MazeE1	6798(463)	6582(447)	96.34(1.00)	96.41(1.28)	5.06(0.38)	5.04(0.33)	5.46(0.40)	5.47(0.52)
MazeE2	1989(187)	1764(226)	98.55(0.78)	98.81(0.95)	3.45(0.63)	3.14(0.47)	6.57(1.10)	6.64(1.09)
MazeF1	40(0)	40(0)	100.00(0.00)	100.00(0.00)	0.98(0.08)	0.99(0.10)	2.41(0.05)	2.41(0.05)
MazeF2	48(0)	48(0)	100.00(0.00)	100.00(0.00)	0.83(0.06)	0.82(0.08)	3.37(0.05)	3.39(0.07)
MazeF3	74(5)	74(6)	100.00(0.00)	100.00(0.00)	0.67(0.08)	0.65(0.17)	4.57(0.07)	4.57(0.07)
MazeF4	186(20)	183(24)	100.00(0.00)	100.00(0.00)	0.97(0.10)	0.92(0.08)	6.34(0.46)	6.30(0.63)
MiyazakiA	4114(357)	4011(405)	96.78(1.64)	95.86(1.22)	4.64(0.31)	4.66(0.32)	4.68(0.30)	4.82(0.25)
MiyazakiB	3295(292)	3267(278)	96.44(1.41)	96.49(1.59)	4.21(0.33)	4.01(0.33)	5.53(0.40)	5.52(0.45)
Woods100	96(9)	94(7)	100.00(0.00)	100.00(0.00)	0.44(0.06)	0.43(0.05)	3.23(0.35)	3.45(1.44)
Woods101	465(48)	452(47)	100.00(0.00)	100.00(0.00)	2.91(0.41)	3.13(0.63)	5.60(0.36)	5.81(0.40)
Woods101.5	806(73)	818(59)	100.00(0.00)	99.68(0.82)	3.84(0.95)	3.79(0.81)	6.33(0.52)	6.21(0.44)
Woods102	1756(172)	1655(180)	99.95(0.25)	100.00(0.00)	4.03(0.32)	3.81(0.31)	7.12(0.60)	7.42(0.83)
Woods14	149(3)	158(7)	100.00(0.00)	100.00(0.00)	0.44(0.21)	0.75(0.21)	12.74(0.22)	13.21(1.55)

Table 1. Means and standard deviations (in brackets) of collected metrics for each maze and each version of BEACS: overall, there were no major statistical differences between the two versions. The table is sorted by mazes in alphabetical order. Complexities of the mazes come from [1]. ∅ indicates that no values were provided because corresponding mazes were not aliased ones: perceptions of each state are unique. Means and standard deviations in orange (resp. in red) correspond to a p-value inferior to the significance threshold 0.05 (resp. inferior to 0.01).

to the knowledge ratio. All these averages were compared using the Welch t-test with (Welch-) Satterthwaite degrees of freedom (significance threshold 0.05).

Those metrics were chosen to enable a comparison of both the environmental representations (sizes of populations, knowledge ratios and average EP-accumulated errors) and the decision policies (average numbers of actions required to reach the exit) built by the agents.

3.3. Performance

Table 1 summarizes all results achieved by BEACS and its debiased version. In general, there are no statistical differences in the classifier population sizes in 19 of 23 mazes, the knowledge ratios achieved in 21 of 23 mazes, the average EP-accumulated errors in 17 of 23 mazes, and the averaged numbers of actions needed to reach the exit in 20 of 23 mazes. d-BEACS has smaller classifier populations than those of BEACS in Maze10 ($p = 0.012$), MazeE2 ($p = 9.81e{-}5$) and Woods102 ($p = 0.030$), while they are larger in Woods14 ($p = 2.95e{-}8$). The knowledge ratios achieved by d-BEACS are smaller than those of BEACS in MiyazakiA ($p = 0.016$) and Woods101.5 ($p = 0.043$). The average EP-accumulated errors of d-BEACS are smaller than those of BEACS in Maze10 ($p = 0.036$), MazeE2 ($p = 0.036$), MazeF4 ($p = 0.024$), MiyazakiB ($p = 0.021$), Woods102 ($p = 8.98e{-}3$), while they are larger in Woods14 ($p = 4.49e{-}7$). The averaged numbers of steps needed by d-BEACS to reach the exit are larger than those of BEACS in Maze10 ($p = 0.032$), MiyazakiA ($p = 0.044$) and Woods101 ($p = 0.031$).

When there is a statistical difference, the removal of the bias enables d-BEACS to build more compact classifier populations that are also more consistent with the environments. But, it comes at the cost of slower task resolution and some environmental transitions that are still poorly known or experienced. These few differences were expected because the removed bias favored the building of decision policies and selection of actions based on perceptible consequences. This bias prevented agents, for example, from choosing actions that would lead them bump into a wall, making it easier for them to build decision policies that ignore such actions. As well, in environmental settings where changes are not easily perceptible, such as a long corridor or the crossing of a desert, this bias altered the selection

of actions during exploration and, therefore, the computation of probabilities related to the occurrences of learned transitions.

In general, this learning bias does not affect BEACS performance in mazes, neither environmental representations nor decision policies. Since BEACS is based on ACS2, it is also possible to extend this result to all ALCS with this bias. Further analysis is still possible and could be the subject of future research. It could be interesting to analyze the extent to which the characteristics of the labyrinths have an impact on ALCS learning, whether in terms of building an environmental representation or a decision-making policy. For example, Woods14 is challenging because its solving implies the ability to find and stabilize long action chains, especially in non-deterministic set-up.

4. Solving the Cart-Pole Problem

Quite surprisingly, to the best of our knowledge, the classic cart-pole problem has never been solved by any ALCS. The only attempt found in the literature does not meet the requirements for solving [12]. We then explain why this classic problem has not been solved until today.

4.1. Hypothesis to solve the cart-pole problem with ALCS

The cart-pole problem consists of controlling a cart that can only move on a horizontal rail [2]. A vertical pole is attached to this cart by a free joint. By moving only the cart to the left or right, the objective is to keep the pole in a vertical position for as long as possible. The whole problem is described by four real values corresponding to cart position, cart velocity, pole angle, and pole angular velocity. The pole is upright on the cart at the beginning. An agent is given a reward of +1 for every step taken as long as the pole does not fall. The problem is considered solved when the average reward received by the agent is greater than or equal to 475 over the last 100 trials to keep the pole upright.

This problem remained unsolved for ALCS for two main reasons.

Firstly, ALCS were explicitly designed to learn in evolving environments, and thus promote actions that lead to changes: this is the learning bias we discussed in the previous section. However, this bias may be counterproductive in solving the cart-pole problem, as the objective is not to see the pole fall by maintaining the initial state as long as possible.

Secondly, ALCS were not designed to handle real-value inputs. Letting ALCS directly learn from the real-value inputs makes them build populations of classifiers that will grow infinitely, thus preventing them from learning and solving their task. Discretization is one technique that enables ALCS to learn and solve tasks with real values. As ALCS learn by correctly anticipating consequences of selected actions, discretization makes perceived situations non-deterministic: a given action for a discretized state may have different consequences. Using discretization then implies enabling ALCS to correctly anticipate all the next reachable discretized states for each situation. In other words, ALCS need a mechanism to handle this kind of non-determinism due to discretization.

We therefore hypothesize that an ALCS such as the debiased version of BEACS (d-BEACS) can solve the cart-pole problem because the learning of d-BEACS is not concerned with the previous two reasons that make the problem unsolved for ALCS.

4.2. Experimental protocol and metrics

Using the CartPole-v1 environment from the gymnasium API [23], performance of d-BEACS, BEACS, ACS2 and its related debiased version d-ACS2 are compared in an experimental protocol set to address the following question: **Can an ALCS solve the cart-pole problem if it is not influenced by the learning bias that promote actions leading to changes and if it can handle non-determinism due to discretization?**

We decided to use these four ALCS because :

- ACS2 is the ALCS behind almost all the others. This ALCS is then used as a control experiment as it is influenced by the learning bias and it cannot handle non-determinism due to discretization.
- d-ACS2, the debiased version of ACS2, cannot handle non-determinism. This ALCS is then used to see whether the bias removal helps solve the cart-pole problem.

- BEACS is an ALCS that can deal with non-determinism. This ALCS is then used to see if this capability alone is necessary to solve the cart-pole problem.
- d-BEACS is the ALCS that we expect to solve the cart-pole problem as it is not influenced by the learning bias and it can handle non-determinism.

For each ALCS, 30 runs were performed. A run firstly consists of a succession 500 episodes that are constrained exploration of the environment until the maximum length of episode (500) is reached, the center of the cart reaches the edge of the display (the cart position is greater than ±2.4) or the pole falls (the pole angle is greater than ±12°). The parameters of the ALCS are for these first 500 episodes: the ϵ-greedy policy used to select actions is set with $\epsilon = 0.5$; the learning rate of the Anticipatory Learning Process is set to 0.05; the maximal number of actions in classifiers, is set to 1; the genetic generalization component is deactivated by setting $\mu = 0$ and $\chi = 0$; the learning rate of the Reinforcement Learning component is set to 0.1, and the discount factor is set to $\gamma = 0.99$. Then, the ALCS are switched to pure exploitation (*i.e.* without Anticipatory Learning Process) during 500 episodes where the parameters are unchanged except the ϵ-greedy policy with $\epsilon = 0$. In each run, the length of each episode is recorded and averaged over the last 100 episodes to check whether the problem is considered solved or not. Other ALCS-related parameters not described here are initialized to the default values provided in [7] for ACS2 and in [20] for BEACS.

We have decided not to use the genetic generalization mechanism, as the objective of this experimental protocol is not to study the population of classifiers *per se*. Proving that ALCS is capable of solving this type of reinforcement learning is a first step on which we focus, before trying to find how to achieve the best performance with a set of maximally general and accurate classifiers (which could be a future research topic). Classifiers with multiple actions are dedicated to bridge distinct states that are indistinguishable by an agent. Once again, the aim of this task is for an agent to maintain a predefined state: using such classifiers could then alleviate the performance of ALCS, which is not desired here. We finally decided to use a discount factor of 0.99 to put more emphasis on long-term benefits and a learning rate β of 0.1 to speed up learning, although the reward predictions made could be noisier.

Concerning the real-value inputs of the cart-pole problem, we decided to ignore the cart position and velocity, as the cart-pole problem can be solved using only the pole angle and angular velocity. The range of values of the pole angle has been divided into 6 uniform buckets. The range of values of the pole angular velocity is unbound. Observing the values perceived by an agent when collecting data and experimentally playing with the environment, most of the values perceived were bound between $[-1; 1]$: we have then divided this range into 3 uniform buckets, where the values observed outside this range have been associated with the nearest bucket. This discretization has been chosen to give the pole angle more importance than the pole angular velocity while maintaining a limited observation space.

4.3. Performance

As depicted in the figure 1, d-BEACS is the only ALCS that has been able to solve the cart-pole problem, consistently achieving an average episode length of 500 around the 600 episodes. More precisely, this system has solved

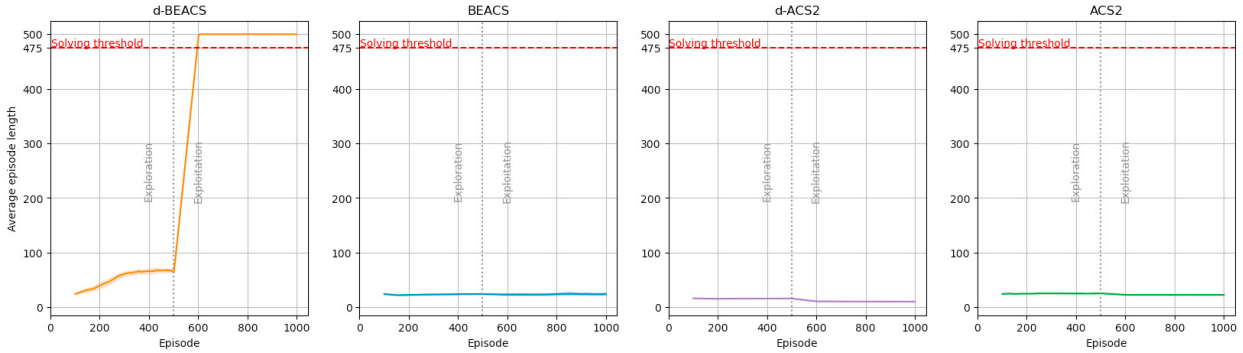

Fig. 1. Performance of BEACS and its debiased version d-BEACS in the Cart-Pole environment, along with the ones of ACS2 and its debiased version d-ACS2. The plotted lines depict the achieved averaged episode length over the last 100 episodes. d-BEACS is the only ALCS able to consistently solve the cart-pole problem.

the cart-pole problem in all of its 30 runs, while the other three have failed to solve it in all of their runs. As the four ALCS use the same set of parameters, the differences in results come from the presence of the learning bias and their ability to anticipate all the next reachable discretized states for each situation. Thus, these results show that ALCS can solve the cart-pole problem, provided that they can handle non-determinism due to discretization and they are not influenced by the learning bias that promotes actions leading to changes.

Here, our objective is not to find the best way for ALCS to solve the cart-pole problem, but to show that it can be solved by these systems. So future research work can now focus more precisely on how these systems can solve this problem or other classic reinforcement problems like the mountain car. It is also possible to focus more on the ALCS parameters that may impact on the resolution of these tasks, on how the chosen discretization influences both the building of the classifiers and the resolution of the tasks, or on the built classifier populations themselves.

5. Ontology as a ”safeguard” for ALCS

Neurosymbolic AI aims to combine the strengths of inductive learning, as found in neural networks or rule-based learning systems like ALCS, with the deductive reasoning capabilities of symbolic knowledge representations such as ontologies [8]. This hybrid paradigm leverages adaptability and generalization power of a learning system while benefiting from the transparency, interpretability, and enforcement of restrictions provided by symbolic knowledge [9]. We then propose a new framework between ALCS and Ontologies that, to the best of our knowledge, has not been proposed in the literature.

5.1. Description of the proposed position

ALCS function as inductive agents that learn their condition-action-effect classifiers from experience. However, their inductive nature makes them potentially prone to invalid decisions, especially in environments that require expert-level understanding, safety guarantees, or consideration of domain-specific constraints. To address this, a solution consists of coupling ALCS with an ontology, which acts as a *“semantic safeguard”*. This ontology serves as a reasoning layer that supervises the ALCS, helping it avoid dangerous or inefficient actions and promoting those that align with specific domain knowledge or some user preferences. Semantic Web Rule Language (SWRL) rules are used within the ontology to formally encode such knowledge and constraints, enabling the system to infer recommendations, reject unsafe actions, or validate the agent’s choices based on symbolic reasoning [11].

An interface between ontologies and ALCS is needed to make these systems communicate with each other. Setting up such an interface is facilitated by the fact that ALCS use symbolic representations to carry out their learning. An ALCS could suggest to this interface actions they want to carry out or classifiers they want to use. This interface can then feed these elements into an ontology, which can request additional information before returning a semantic answer. Thus, the ontology helps guide the agent toward higher-quality behaviors by influencing the selection of actions whose outcomes are more likely to yield positive reinforcement and consistent with its encoded knowledge.

More precisely, before executing an action proposed by an ALCS, the ALCS consults the ontology through this interface by transmitting structured descriptions of its current context to the ontology. These descriptions may include the agent’s symbolic perception of its current position, the action it intends to take, the consequences it anticipates, and optionally internal variables related to the classifiers that match the current perception and the selected action. The ontology then receives these descriptions that enable it to *validate* or *reject* the proposed action. For example, in the context of mazes, the ontology may reject actions that would lead the agent into a wall, a dangerous zone, or an area that violates a domain constraint. It may also *suggest alternative actions* that are considered safer or more efficient.

To fulfill its role as a semantic safeguard, the ontology returns a structured response that guides the agent’s behavior. Specifically, the ontology’s output includes two key fields: `"validation"` and `"reason"`. The `"validation"` field related to the proposed action can take one of four values:

- `valid`: the action is accepted and aligns with the ontology’s symbolic preferences;
- `preferred`: the action is considered the best choice according to symbolic reasoning;
- `rejected`: the action is forbidden or dangerous, as it violates symbolic constraints;
- `suggestion`: the action is not valid, but an alternative is provided in an additional field `"proposition"`.

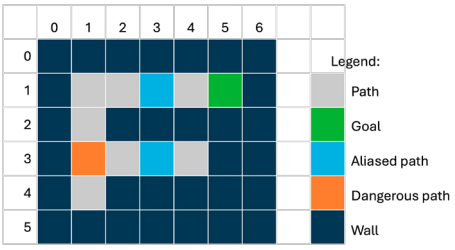

Fig. 2. Adapted maze F4 used in the examples of our neurosymbolic approach. Agents evolving in this maze only perceive the eight cells around their current position, starting from North and clockwise. Cyan cells are aliased because an agent perceives those different cells in the same way. The orange cell is a particular cell whose danger is invisible for an agent as it gets no clues about it (no perceptive input or negative reward). Cell (x, y) is related to the x-th column and the y-th line: $(5, 1)$ is the goal cell, for example.

The `"reason"` field provides a human-readable justification, allowing the user to understand the symbolic reasoning behind the decision. This contributes to the overall explainability and transparency of the decision-making process.

To better understand how symbolic reasoning supports or constrains the ALCS learning process, we provide two examples based on the maze shown in figure 2, along with the corresponding ontology rules and responses. The goal of our agent is to build a representation of the maze by exploring the latter in both examples. But it must avoid dangerous cells at any cost, and it should favor exploration of the less visited cells: this specific knowledge is encoded within the associated ontology.

5.2. An example of rejection, validation and preferred cases

Let us suppose that the agent is located at position $(1, 2)$ and these two classifiers that match this position:

```
Classifier 1:
C: #.#.Wall.#.Path.#.#.#
A: Move to North-East
E: (Wall.Wall.Path.Wall.Wall.Path.Path.#)
Experience: 450
```

```
Classifier 2:
C: #.#.Wall.#.Path.#.#.#
A: Move to North
E: (Wall.Wall.Path.Wall.#.#.#.#)
Experience: 150
```

Given this position $(1, 2)$, the ALCS action selection policy chooses to move to North and submits this action to its associated ontology. Through the interface, the ontology asks for a classifier that matches this position and this action to check if it leads to a cell that is walkable and not dangerous. `Classifier 1` is chosen by the ALCS and is submitted to the ontology. To check if `Classifier 1` is safe, the ontology uses the following SWRL-like meta-rule :

```
computeAnticipatedCell(?currentPerception, ?classifier, ?anticipatedCell),
isPath(?anticipatedCell), not(isDangerous(?anticipatedCell))
→ isClassifierSafe(?currentPerception, ?classifier)
```

The ontology asks then the ALCS for another action with the corresponding classifier as the first rule is checked. The ALCS selects a second action and submits `Classifier 2`, that is also safe. As a consequence, a SWRL-like meta-rule is used to find out if the initial intended action is consistent with promoting exploration of less-visited cells:

```
experienceCount(?clf1, ?exp1), experienceCount(?clf2, ?exp2),
swrlb:lessThan(?exp1, ?exp2)
→ isExplorationPromising(?clf1, ?clf2)
```

Based on these rules, the ontology confirms the classifier proposing to move North (to cell $(2, 1)$) as the most suitable choice: it leads to a free, non-dangerous, and less visited cell. The ontology then returns the following response, enabling the ALCS to carry out the intended action:

```
{   "validation": "preferred",
    "reason": "The selected action leads to a free, safe, and less visited cell
               that promotes exploration of the environment." }
```

We can also consider the case where the initial proposed action is to go North-East, and these same classifiers are submitted to the ontology. Depending on whether the promotion of exploration is a hard or soft constraint, the ontology could then have respectively refused (and even suggested going North) or validated the action.

5.3. An example with a violated constraint and a suggestion

Now, the agent is located at position $(1, 4)$ and intends to go North. Through the interface, the ontology asks for a classifier that matches this position and this action. The ALCS sends the following classifier being the most fitted to its environment to the ontology:

```
Classifier 3:
C: Path.#.Wall.Wall.#.#.#.#
A: Move to North
E: (#.Wall.Path.#.Path.#.#.#)
Experience: 250
```

The ontology uses symbolic reasoning to first detect if the classifier is safe, using the previous SWRL-like meta rule `isClassifierSafe`. However, according to the ontology, the cell $(1, 3)$ is marked as dangerous, leading `Classifier 3` to be deemed unsafe. The ontology tries to find an alternative, like another action leading to a free and safe cell, such as $(2, 3)$. For this purpose, the ontology requests different classifiers matching the current cell $(1, 4)$. It enables the ontology to use this rule to find an action to suggest aligned with the ontology's preferences:

```
findSafeClassifiers(?currentPerception, ?classifiers, ?safeClassifiers),
findExplorationPromisingClassifier(?safeClassifiers, ?classifier)
→ suggestAction(?currentPerception, ?classifiers)
```

Based on these rules, the ontology rejects the classifier leading North and suggests going North-East, which is safer and acceptable in terms of symbolic reasoning. The ontology returns the following response:

```
{   "validation": "suggestion",
    "proposition": "Move to North-East",
    "reason": "Going North is dangerous as the anticipated cell is marked as
               dangerous. Consider moving to a safer adjacent cell which is also
               exploration promising." }
```

Finally, the interface between the ALCS and the ontology enables the ALCS to adapt its decision to comply with the domain-knowledge constraints. If ontology can't make a suggestion, the ontology can return a rejection response and let the agent propose a new action. This example also shows that other rules embedded in the ontology may be necessary to prevent the agent from getting stuck, or have to be carefully designed: if all actions are rejected in a given situation, the agent must still continue learning. This example illustrates how the ontology prevents risky behavior and guides the agent toward viable, safer decisions.

6. Conclusion

In this paper, we have addressed an important limitation of ALCS: their learning bias favoring perceptible changes. We have also shown through experiments that ALCS can, for the first time, successfully solve the cart-pole problem. We finally introduced a neurosymbolic approach in which ALCS are coupled with ontologies acting as semantic safeguards. Through two illustrative examples, we showed how ontological reasoning can be used to validate, reject, or adjust the agent's decisions, thereby contributing to make its behavior safer and better guided by domain knowledge.

Future work will focus on implementing and evaluating a fully operational ALCS–Ontology interface in more complex environments where domain constraints, safety requirements, and partial observability play a critical role, such as in robotic surgery. This implementation will allow us to assess the robustness and adaptability of our approach in more realistic and high-stakes scenarios.

In such contexts, where expert supervision and domain-specific rules are essential, the proposed coupling ALCS–Ontology also opens the door to human-in-the-loop interaction. Domain experts can inject, modify, or refine ontological rules to influence agent behavior in a declarative and interpretable way, without retraining or manually editing classifier rules (which remains possible). Those ontological rules can be domain- or application-specific as well as related to the way learning is performed. This makes the system not only more explainable, but also more controllable and collaborative. The expert becomes an active participant in the learning loop, understanding and guiding behavior through knowledge.

References

[1] Bagnall, A.J., Zatuchna, Z.V., 2005. On the classification of maze problems, in: Foundations of Learning Classifier Systems. Springer, pp. 305–316.

[2] Barto, A.G., Sutton, R.S., Anderson, C.W., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics SMC-13, 834–846.

[3] Butz, A.M.V., Goldberg, B.D.E., Stolzmann, C.W., 2002a. The anticipatory classifier system and genetic generalization. Natural Computing , 427–467.

[4] Butz, M.V., 2001. Biasing exploration in an anticipatory learning classifier system, in: International Workshop on Learning Classifier Systems, pp. 3–22.

[5] Butz, M.V., Goldberg, D.E., Stolzmann, W., 2000. Probability-enhanced predictions in the anticipatory classifier system, in: International Workshop on Learning Classifier Systems, pp. 37–51.

[6] Butz, M.V., Goldberg, D.E., Stolzmann, W., 2002b. Anticipatory learning classifier systems. Genetic Programming and Evolvable Machines 3, 111–133.

[7] Butz, M.V., Stolzmann, W., 2001. An algorithmic description of acs2, in: International Workshop on Learning Classifier Systems, pp. 211–229.

[8] Garcez, A.d., Besold, T.R., Lamb, L.C., Serafini, L., Spranger, M., 2019. Neurosymbolic learning systems: The future of artificial intelligence. IEEE Intelligent Systems 34, 17–26.

[9] Garcez, A.d., Lamb, L.C., 2023. Neurosymbolic ai: The 3 rd wave. Artificial Intelligence Review 56, 12387–12406.

[10] Hoffmann, J., 2003. Anticipatory behavioral control, in: Anticipatory behavior in adaptive learning systems. Springer, pp. 44–65.

[11] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M., et al., 2004. Swrl: A semantic web rule language combining owl and ruleml. W3C Member submission 21, 1–31.

[12] Kozlowski, N., Unold, O., 2020. Investigating exploration techniques for acs in discretized real-valued environments, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pp. 1765–1773.

[13] Kozłowski, N., Unold, O., 2021. Anticipatory classifier system with average reward criterion in discretized multi-step environments. Applied Sciences 11, 1098.

[14] Kozlowski, N., Unold, O., 2022. Internalizing knowledge for anticipatory classifier systems in discretized real-valued environments. IEEE Access 10, 33816–33828.

[15] Lisi, F.A., Straccia, U., 2022. Neural-symbolic learning and reasoning: A survey and interpretation. Artificial Intelligence 311, 103737.

[16] Métivier, M., Lattaud, C., 2002. Anticipatory classifier system using behavioral sequences in non-markov environments, in: International Workshop on Learning Classifier Systems, pp. 143–162.

[17] Orhand, R., Collet, P., Parrend, P., Jeannin-Girardon, A., 2023. Cracs: Compaction of rules in anticipatory classifier systems, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, pp. 1838–1845.

[18] Orhand, R., Jeannin-Girardon, A., Parrend, P., Collet, P., 2020a. Bacs: A thorough study of using behavioral sequences in acs2, in: International Conference on Parallel Problem Solving from Nature, Springer. pp. 524–538.

[19] Orhand, R., Jeannin-Girardon, A., Parrend, P., Collet, P., 2020b. Pepacs: Integrating probability-enhanced predictions to acs2, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, p. 1774–1781.

[20] Orhand, R., Jeannin-Girardon, A., Parrend, P., Collet, P., 2022. Accurate and interpretable representations of environments with anticipatory learning classifier systems, in: European Conference on Genetic Programming (Part of EvoStar), Springer. pp. 245–261.

[21] Smierzchała, Ł., Kozłowski, N., Unold, O., 2023. Anticipatory classifier system with episode-based experience replay. IEEE Access 11, 41190–41204.

[22] Stolzmann, W., 1999. An introduction to anticipatory classifier systems, in: International Workshop on Learning Classifier Systems, Springer. pp. 175–194.

[23] Towers, M., Kwiatkowski, A., Terry, J., Balis, J.U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al., 2024. Gymnasium: A standard interface for reinforcement learning environments. arXiv preprint arXiv:2407.17032 .

[24] Unold, O., Rogula, E., Kozłowski, N., 2019. Introducing action planning to the anticipatory classifier system acs2, in: International Conference on Computer Recognition Systems, Springer. pp. 264–275.