Towards the Integration of Reinforcement Learning into MASPY

Alexandre L. L. Mellado[1],
André Pinz Borges[1], **Rafael C. Cardoso**[2], **Gleifer Vaz Alves**[1]

[1]Federal University of Technology Paraná (UTFPR)
Ponta Grossa, PR, Brazil

[2]University of Aberdeen
Aberdeen – United Kingdom

mellado@alunos.utfpr.edu.br

{apborges, gleifer}@utfpr.edu.br

rafael.cardoso@abdn.ac.uk

Abstract. *Learning in symbolic agent architectures remains a key challenge in the development of adaptive multi-agent systems. This paper introduces a learning module for MASPY, a Python-based framework inspired by the Belief-Desire-Intention (BDI) model. The module enables agents to learn optimal actions using tabular reinforcement learning algorithms, such as Q-Learning and SARSA. To support this, we propose the SART methodology, which decomposes the learning environment into four structured components: States, Actions, Rewards, and Transitions. This structure allows MASPY agents to perceive their environment through defined percepts, act through decorated functions, and adapt over time using discrete learning strategies. The learning module offers a unified Python-based architecture for symbolic reasoning agents that learn through reinforcement training. This is shown practically with a toy problem where agents are able to learn to execute the actions of a previously unknown environment.*

1. Introduction

The integration of learning capabilities into multi-agent systems (MAS) has become increasingly important for enabling intelligent and adaptive behavior in dynamic environments [Zhu et al. 2024, Hu et al. 2024]. Traditional agent architectures, particularly those based on the Belief-Desire-Intention (BDI) model [Bratman 1987], offer strong frameworks for symbolic reasoning and goal-driven action, but often lack mechanisms for autonomous learning from experience. Reinforcement learning (RL), on the other hand, provides a well-established paradigm for enabling agents to optimize behavior through trial-and-error interaction with an environment [Sutton and Barto 2018].

This work introduces a modular learning component within MASPY [Mellado et al. 2023][1], a Python-based framework for the development of BDI Multi-Agent Systems. The proposed module equips each agent with the ability to learn optimal behaviors using tabular RL algorithms such as Q-Learning and

[1]https://github.com/laca-is/MASPY

SARSA [Sutton and Barto 2018]. This paper also presents the design of the SART methodology, that works by structuring the environment according to States, Actions, Rewards, and Transitions and enables seamless integration between symbolic agent reasoning and symbolic learning strategies.

MASPY's learning module is designed to be an internal element of the agent-environment interface. It leverages structured percepts and annotated actions to construct a discrete model of the environment and autonomously derive policies that improve decision-making over time. The organization of this paper follows by first presenting some related works in section 2; Describing the SART methodology and how it is implemented in the MASPY framework in section 3; Illustrate a practical MASPY example, through SART to train it in section 4; and finally a conclusion on section 5.

2. Related Work

Integrating RL into symbolic multi-agent systems has attracted significant research attention in recent years. For instance, Jason-RL [Bosello and Ricci 2019] integrates RL in their BDI-based agent reasoning model, providing them ways to execute some plans according to their BDI logic and some learned by using RL.

[Patrascu 2025] proposed a constructive symbolic RL architecture using intuitionistic logic to guarantee correct policies, while [Sarathy et al. 2020] extended symbolic planners with RL capabilities to discover missing operators dynamically. [Subramanian et al. 2024] designed a neuro-symbolic multi-agent system using first-order logic to coordinate learning among multiple agents.

Other hybrid frameworks have attempted to blend symbolic reasoning with neural policy learning. [Shindo et al. 2025] introduced BlendRL, combining symbolic reasoning layers with neural reinforcement modules, whereas [Zou et al. 2024] applied RL-enhanced symbolic theorem proving to geometric reasoning tasks.

While these works successfully advance hybrid reasoning and learning, they typically focus on domain-specific or specialized neuro-symbolic approaches and none offers a BDI reasoning as the symbolic component. In contrast, MASPY offers a general-purpose, Python-native framework that integrates symbolic BDI reasoning with discrete RL capabilities. By leveraging the SART methodology, the learning module in MASPY enables agents to derive optimal policies grounded in structured perceptual states and formally defined transition functions, providing a lightweight yet extensible solution for integrating learning into cognitive agent architectures.

3. SART Methodology

The MASPY framework adopts a structured approach to integrating learning into multi-agent systems through the SART methodology, which decomposes the environment into four fundamental components: **States**, **Actions**, **Rewards**, and **Transitions**. This model formalizes agent-environment interaction in a manner suitable for RL.

- **States**: Define all possible configurations of the environment.
- **Actions**: Define the operations the agent can perform in with the environment.
- **Rewards**: Assign numerical values to evaluate the outcome of actions on states.
- **Transitions**: Describe how an action changes the state and produces a reward.

3.1. States

States define the complete configuration of the environment at a given moment, expressed as combinations of values across all relevant contextual information. Each state represents a complete snapshot of the environment [Sutton and Barto 2018].

The state space S is partitioned into three subsets, S_0: the set of Initial States; S_i: the set of Intermediate States; and S_n: the set of Final States.

In MASPY, States are represented using Percepts. These abstract meaningful environmental features. A state is defined as the combination of values across all active percepts. They are characterized by a name, a finite set of possible values, and a grouping strategy that defines its structural semantics.

- **Definition**: `Percept(<Name>, <Values>, <Group>)`
  - **Name**: Identifier of the percept (must be a string).
  - **Values**: Possible values for the percept (list, tuple, or set).
  - **Group**: Determines how the percept values are interpreted.

3.2. Actions

Actions represent the operational interface through which the agent affects their environment. Each action is associated with a defined group of permissible values and a transition function that maps the current state and action parameters to a resulting state [Sutton and Barto 2018].

These actions A are determined by decorators applied to functions within the MASPY environment class. Each decorator assigns semantic and functional information to an action.

- **Definition**: `@action(<Group>, <Values>, <Transition>)`
  - **Group**: Categorizes the interpretation of action values.
  - **Values**: Set of allowed values for the action (literal or dynamic).
  - **Transition**: A function that modifies the environment state and returns the resulting state and reward.

3.3. Rewards and Transitions

Rewards R quantify the desirability of transitions, assigning scalar values to outcomes of state-action pairs. Positive rewards incentivize behavior, while negative values discourage it. Transitions T, deterministic or stochastic, specify the evolution of the environment in response to agent actions, forming the basis for policy optimization [Sutton and Barto 2018]. Each transition is then represented as:

$$T = \{(s, a) \to (s', r)\},$$
$$\text{where } \{s, s'\} \in S, a \in A, and\ r \in R$$

This reward system underpins RL by helping the agent develop and refine optimal strategies. Positive values encourage actions; Negative values discourage actions; and Zero values represent neutral feedback.

Transitions define the causal relationship between actions and changes in the environment state, and the rewards this transition brings. Each transition is implemented as a function that receives the current state and action parameters; Computes the resulting state; and returns the new state and the associated reward.

3.4. MASPY Learning Module

MASPY is a framework comprised already of four classes, agents for BDI reasoning, environment for perceptions, communication for exchanging information, and the admin for system management. The Learning module is then the fifth class added with the object to extend agent's reasoning by training in given environments.

The MASPY learning module operationalizes this methodology by automatically extracting state and action definitions from a given environment specification, constructing a discrete, learning-ready model by identifying percepts and transition-annotated functions. The following grouping semantics are supported for both states and actions:

- **listed**: treats values as a flat list.
- **cartesian**: produces a Cartesian product of value ranges.
- **sequence**: generates a consecutive range between two numeric bounds.
- **combination**: considers all unordered subsets of the value set.
- **permutation**: generates all ordered arrangements of the value set.

Only percepts and actions that conform to the required structure are included in the learning model. Transition functions must explicitly return the resulting state and its associated reward. The module currently supports learning over discrete state-action spaces using tabular RL algorithms, specifically Q-Learning and SARSA. It produces a learned policy represented as a mapping from states to optimal actions, enabling the agent to act adaptively within complex, domain-specific environments.

4. Illustrative MASPY Example Using SART

In this example, we consider a robot agent that must allocate objects that are on a shelf into specific boxes. Figure 1 shows the system diagram with the environment for allocating objects and the agent that will act on this environment. The environment in question requires two perceptions, one for each object that contains where it is currently allocated and the action of moving an object to a given location.

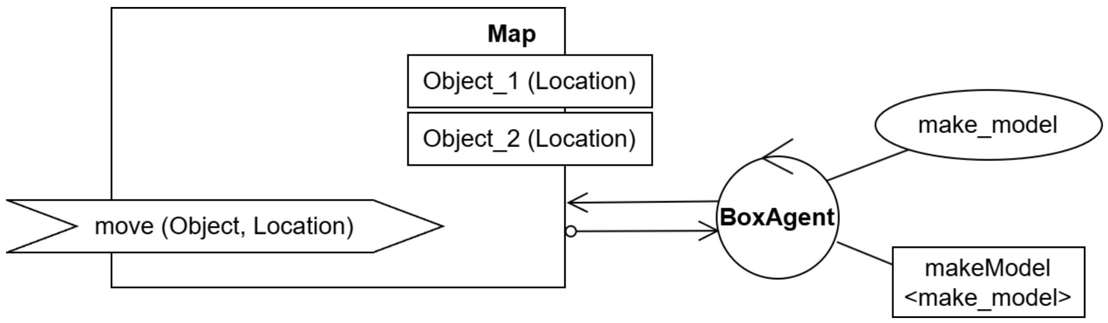

Figure 1. Modelling of the Example

To demonstrate the application of the SART methodology, we present a simplified environment in which an agent must allocate two objects to target locations. The model follows the SART structure: design of **States**, **Actions**, **Rewards**, and **Transitions**.

4.1. States

The state of the environment is defined by the positions of two distinct objects, o_1 and o_2. Each object may be located either on a shelf (*Shelf*) or in one of two boxes (*Box$_1$*, *Box$_2$*). Thus, each state is a tuple representing the current positions of both objects. We first define the states for each category:

- S_0: Initial state where both objects are on the shelf.
  $S_0 = \{[(o_1, \text{Shelf}), (o_2, \text{Shelf})]\}$
- S_i: Intermediate states where one object has been moved.
  $S_i = \{[(o_1, \text{Box}_1), (o_2, \text{Shelf})], [(o_1, \text{Box}_2), (o_2, \text{Shelf})], \ldots\}$
- S_n: Final states where both objects are located in boxes.
  $S_n = \{[(o_1, \text{Box}_1), (o_2, \text{Box}_1)], [(o_1, \text{Box}_2), (o_2, \text{Box}_2)], \ldots\}$

The snippet in Code 1 presents how this is then translated in MASPY. Each percept that is meant to be part of the state is set with its name, possible values and appropriate group. And finally, the initial state is explicitly declared as the possible start.

```
self.create(Percept("Object_1",("Shelf","Box_1","Box_2"),listed))
self.create(Percept("Object_2",("Shelf","Box_1","Box_2"),listed))
self.possible_starts = {"Object_1": "Shelf", "Object_2": "Shelf"}
```

Code Listing 1. State Percepts

4.2. Actions

The action space A consists of four discrete operations, each corresponding to moving a specific object to a designated box.

$$\begin{aligned} A = \{a_1 &= \text{Move } o_1 \text{ to Box}_1, \quad a_2 = \text{Move } o_1 \text{ to Box}_2, \\ a_3 &= \text{Move } o_2 \text{ to Box}_1, \quad a_4 = \text{Move } o_2 \text{ to Box}_2\} \end{aligned}$$

These set of actions are translated as a decorator above the appropriate environment function, with its group, values and transition function, seen in Code 2. The Cartesian group is used to combine all objects being moved to any box, including all possible actions available.

```
@action(cartesian, (('Object_1','Object_2'),('Box_1','Box_2')), move_t)
def move(self, agt, obj_to_box: tuple[str, str]):
```

Code Listing 2. Action Decorator

4.3. Rewards

Rewards are used to evaluate the correctness of each allocation. A larger number of rewards could be used for more specification, but the example does not warrant that:

- $r_1 = +5$: Positive reward for placing an object in its designated box.
- $r_2 = -5$: Negative reward for incorrect placement.

4.4. Transitions

Given the current state s, a is the action applied, s' is the resulting state, and r is the reward received. This example contains a transition from the set of all possible transitions:

$$t_2 = ([(o_1, \text{Shelf}), (o_2, \text{Box}_2)], a_1) \rightarrow ([(o_1, \text{Box}_1), (o_2, \text{Box}_2)], r_1)$$

This shows that in a state where o_1 is on the shelf and o_2 is in Box$_2$, applying a_1 (moving o_1 to Box$_1$) transitions it to the state where both objects are in boxes, yielding a reward r_1 of $+5$. All transitions are explicitly enumerated, capturing every valid action-state pair. The transitions are deterministic and reflect the movement logic of the domain.

4.5. Remarks

This example highlights the applicability of the SART methodology to structured symbolic environments. The agent begins with all objects on the Shelf $[(o_1, \text{Shelf}), (o_2, \text{Shelf})]$, with an empty Q-table, and must learn which sequence of actions leads to the objects being placed on their proper boxes, as determined by the reward structure. The environment model is discrete and finite, making it compatible with the tabular Q-Learning algorithm implemented in the MASPY learning module.

In practice, an MASPY agent would consider, during its reasoning, to use the available trained actions, instead of one of their plans, for the execution in this environment. Or these actions can be added to an agent that did not have plans prepared before for this environment, thus extending it beyond its initial capabilities.

The following is the resulting, trained, Q-Table that properly shows how the learned actions in each state to move the objects to their proper places:

(o_1, Pos, o_2, Pos)	a_1	a_2	a_3	a_4
(Shelf, Shelf)	**7.32**	-2.12	-1.72	**5.50**
(Shelf, Box_1)	1.13	-0.92	-1.10	-0.46
(Shelf, Box_2)	**3.74**	-0.48	-0.35	-0.49
(Box_1, Shelf)	-0.57	-0.77	-0.71	**4.52**
(Box_2, Shelf)	-0.67	-1.36	-1.32	1.50

Table 1. Action values for each object position pair

A state-transition diagram representing the full automaton of this environment is provided in Figure 2, where each circle corresponds to a state and each arrow represents an action labeled with the associated reward. These arrows also come with the associated learned value, from the Q-Table. The circle with larger arrow represent the initial state, and concentric circles the final states. The traced arrows represent the learned best paths after training for 1000 episodes in this environment using Q-Learning, and thus the max sum of learned values.

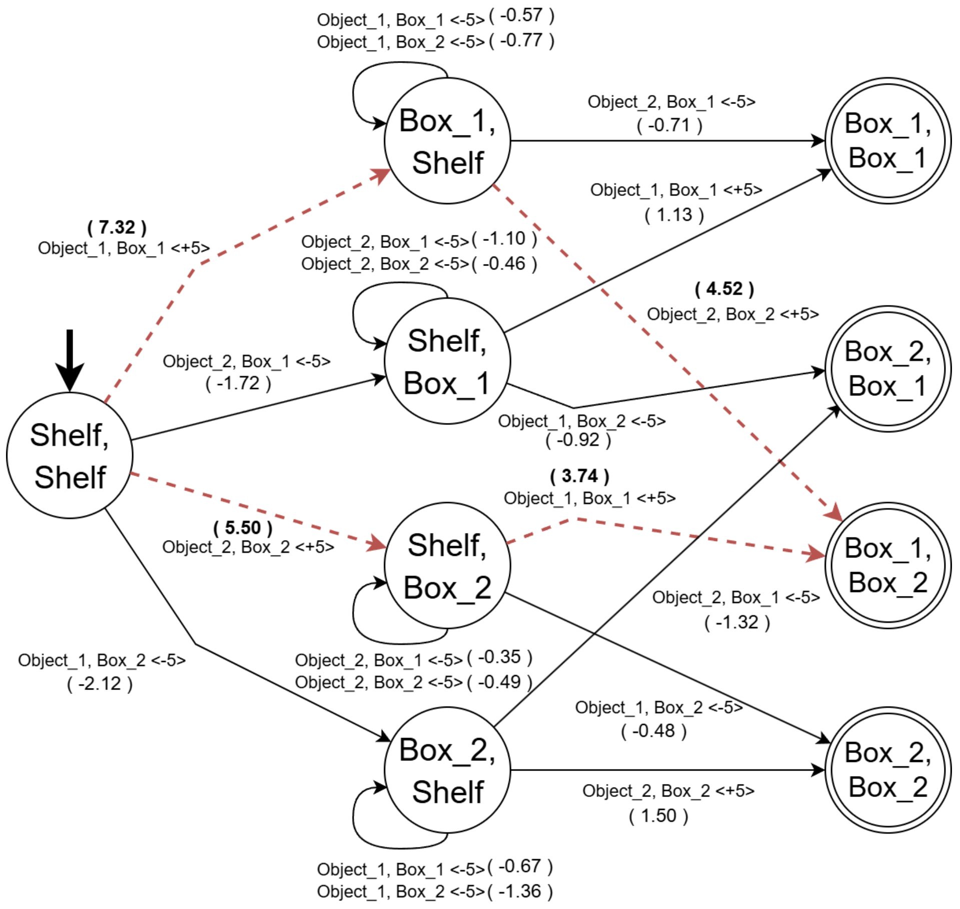
Object_1, Box_1 <-5> (-0.57)
Object_1, Box_2 <-5> (-0.77)
Box_1, Shelf
Object_2, Box_1 <-5> (-0.71)
Box_1, Box_1
Object_1, Box_1 <+5> (1.13)
(7.32) Object_1, Box_1 <+5>
Object_2, Box_1 <-5> (-1.10)
Object_2, Box_2 <-5> (-0.46)
(4.52) Object_2, Box_2 <+5>
Shelf, Box_1
Object_2, Box_1 <-5> (-1.72)
Box_2, Box_1
Shelf, Shelf
Object_1, Box_2 <-5> (-0.92)
(3.74) Object_1, Box_1 <+5>
(5.50) Object_2, Box_2 <+5>
Shelf, Box_2
Box_1, Box_2
Object_2, Box_1 <-5> (-1.32)
Object_1, Box_2 <-5> (-2.12)
Object_2, Box_1 <-5> (-0.35)
Object_2, Box_2 <-5> (-0.49)
Object_1, Box_2 <-5> (-0.48)
Box_2, Shelf
Box_2, Box_2
Object_2, Box_2 <+5> (1.50)
Object_1, Box_1 <-5> (-0.67)
Object_1, Box_2 <-5> (-1.36)

Figure 2. Automaton representation of the state transitions.

5. Conclusion

This paper presented the design and implementation of a learning module for the MASPY framework, enabling agents to improve their behavior through reinforcement learning. By introducing the SART methodology and structuring the environment into States, Actions, Rewards, and Transitions, the proposed system bridges symbolic reasoning and adaptive learning within a unified agent architecture.

Unlike other multi-agent or BDI-based platforms that require external integration for learning, MASPY provides a native mechanism for learning through structured percepts and annotated transitions. The result is a flexible, Python-based system where agents can reason about high-level goals while incrementally learning how to act optimally within their environments.

Future work includes extending the module to support continuous state-action spaces, integrating probabilistic reasoning mechanisms, and evaluating the framework across a broader set of benchmark tasks. MASPY's architecture also opens opportunities for exploring hybrid neuro-symbolic approaches.

Funding: This research is funded by project: 444568/2024-7, CNPq/MCTI/FNDCT Nº 22/2024, *Programa Conhecimento Brasil – Apoio a Projetos em Rede com Pesquisadores Brasileiros no Exterior.*

References

Bosello, M. and Ricci, A. (2019). From programming agents to educating agents - A jason-based framework for integrating learning in the development of cognitive agents. In Dennis, L. A., Bordini, R. H., and Lespérance, Y., editors, *Engineering Multi-Agent Systems - 7th International Workshop, EMAS 2019, Montreal, QC, Canada, May 13-14, 2019, Revised Selected Papers*, volume 12058 of *Lecture Notes in Computer Science*, pages 175–194. Springer.

Bratman, M. (1987). *Intention, Plans, and Practical Reason.* Cambridge: Cambridge, MA: Harvard University Press.

Hu, K., Li, M., Song, Z., Xu, K., Xia, Q., Sun, N., Zhou, P., and Xia, M. (2024). A review of research on reinforcement learning algorithms for multi-agents. *Neurocomputing*, page 128068.

Mellado, A. L. L., G., F. I., Alves, G. V., and Borges, A. P. (2023). Maspy: Towards the creation of bdi multi-agent systems. In *Proceedings of the 17th Workshop-School on Agents, Environments, and Applications (WESAAC 2023)*, pages 106–117.

Patrascu, A. T. (2025). Constructive symbolic reinforcement learning via intuitionistic logic and goal-chaining inference.

Sarathy, V., Kasenberg, D., Goel, S., Sinapov, J., and Scheutz, M. (2020). Spotter: Extending symbolic planning operators through targeted reinforcement learning.

Shindo, H., Delfosse, Q., Dhami, D. S., and Kersting, K. (2025). Blendrl: A framework for merging symbolic and neural policy learning.

Subramanian, C., Liu, M., Khan, N., Lenchner, J., Amarnath, A., Swaminathan, S., Riegel, R., and Gray, A. (2024). A neuro-symbolic approach to multi-agent rl for interpretability and probabilistic decision making.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* MIT press.

Zhu, C., Dastani, M., and Wang, S. (2024). A survey of multi-agent deep reinforcement learning with communication. *Autonomous Agents and Multi-Agent Systems*, 38(1):4.

Zou, J., Zhang, X., He, Y., Zhu, N., and Leng, T. (2024). Fgeo-drl: Deductive reasoning for geometric problems through deep reinforcement learning.